

THE UNIFRAME .NET
WEB SERVICE DISCOVERY SERVICE

TR-CIS-0630-03

Robert W. Berbeco June 27, 2003

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE JUN 2003		2. REPORT TYPE		3. DATES COVERED 00-00-2003 to 00-00-2003	
4. TITLE AND SUBTITLE The Uniframe .Net Web Service Discovery Service				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Indiana University/Purdue University, Department of Computer and Information Sciences, Indianapolis, IN, 46202				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT see report					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 124	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

THE UNIFRAME .NET WEB SERVICE DISCOVERY SERVICE

A Technical Report

Report Number

TR-CIS-0630-03

Submitted to the Faculty

Of Purdue University

By

Robert W. Berbeco

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

June 2003

To Paula.

ACKNOWLEDGMENTS

During the time I spent being a graduate student at the Department of Computer and Information Science, Indiana University-Purdue University-Indianapolis I received support from many different individuals on many levels. My debt of gratitude is spread among my family, other graduate students within the department, and my professors at Indiana University-Purdue University-Indianapolis.

To begin, deep thanks to Professor Rajeev R. Raje, my academic advisor, for his support and willingness to guide me through my graduate program. His advice and knowledge were invaluable, and I truly appreciated his ability to challenge me to always complete my graduate work to the best of my abilities. I have learned a lot from Dr. Raje, both personally and professionally, and what I learned has helped me to grow as a person. These lessons were taken to heart and will be with me for a very long time.

I would like to thank my wife, Paula Berbeco, for her limitless support, patience, understanding, and willingness to listen and respond to me as I worked my way through the coursework for my master's degree and this project.

I would like to thank Professor Yung-Ping Chien and Professor Andrew Olson for taking their time to be on my graduate committee.

Special thanks to Natasha Gupta, a graduate student in the Department of Computer and Information Science program. Her comments and insight were helpful.

I would like to thank the U.S. Department of Defense and the U.S. Office of Naval Research for supporting this research under the award number N00014-01-1-0746.

I also would like to thank the faculty, staff, and other students at the Department of Computer and Information Science for their kindness and conversations during my time spent within the department. I will cherish these memories.

TABLE OF CONTENTS

TABLE OF CONTENTS.....	iv
LIST OF TABLES.....	vi
LIST OF FIGURES.....	vii
ABSTRACT.....	viii
1. INTRODUCTION.....	1
1.1 Motivation.....	2
1.2 Objectives.....	3
1.3 Contributions.....	4
1.4 Organization of this report.....	5
2. BACKGROUND AND RELATED WORK.....	6
2.1 Microsoft .NET.....	6
2.1.1 Internet Information Server (IIS).....	7
2.1.1.1 IIS Security.....	8
2.1.1.2 Virtual Directories.....	9
2.1.2 Web services.....	9
2.3.3 ASP.NET.....	12
2.2 UniFrame.....	14
2.3 Discovery.....	16
2.3.1 Universal Description, Discovery, and Integration (UDDI) Service.....	17
2.3.2 Web Services Inspection Language (WSIL).....	19
2.3.3 Discovery of Web Services (DISCO).....	20
3. ARCHITECTURE.....	23
3.1 URDS Architecture Overview.....	23
3.2 Design Details.....	26
3.2.1 Query Manager (QM).....	26

3.2.2 Headhunters (HH).....	33
3.2.3 Meta-Repository (MR).....	40
3.2.4 Active Registry (AR).....	41
3.3 Implementation.....	41
3.3.1 Technology Used.....	41
3.3.2 Prototype Implementation.....	43
3.3.2.1 Environment.....	44
3.3.2.2 Communication.....	45
3.3.2.3 Security.....	45
3.3.2.4 Programming.....	46
3.3.2.4.1 Service Objects.....	46
3.3.2.4.2 Database Objects.....	49
3.3.2.4.3 User Interface.....	50
4. VALIDATION.....	55
4.1 Experimentations.....	55
4.2 Results.....	56
5. CONCLUSION AND FUTURE WORK.....	63
LIST OF REFERENCES.....	65
APPENDIX.....	67
Source Code.....	67

LIST OF TABLES

Table	Page
Table 3.1 Description of URDS Components (from [4]).....	23
Table 3.2 Data structures used for QM functions.....	28
Table 3.3 Data structures used for Headhunters.....	35
Table 4.1 Number of incoming queries vs. Average Response Time with all HHs.....	57
Table 4.2 Number of Web services vs. Average Response Time.....	59
Table 4.3 Number of HHs vs. Average Response Time.....	60

LIST OF FIGURES

Figure	Page
Figure 2.1 .NET three-tier system (from [15]).....	6
Figure 2.2 A client invoking a Web service via SOAP (from [8]).....	10
Figure 2.3 ASP and ASP.NET Architecture.....	12
Figure 2.4 .NET Architecture (from [14]).....	13
Figure 2.5 URDS Architecture (from [4]).....	15
Figure 2.6 businessEntity structure (from [9]).....	18
Figure 3.1 URDS architecture (from [4]).....	25
Figure 3.2 UNWSDS Implementation.....	44
Figure 3.3 Class Diagrams for the UNWSDS Services.....	46
Figure 3.4 Class Diagrams for the Domain Services.....	48
Figure 3.5 Class Diagrams for Database Objects.....	49
Figure 3.6 UniFrameIndex.aspx.....	51
Figure 3.7 UniFrameQuery.htm.....	51
Figure 3.8 ComponentList.aspx.....	52
Figure 3.9 GetCustomerAccountsClient.aspx.....	52
Figure 3.10 GetInventoryAccountsClient.aspx.....	53
Figure 3.11 BankDataSvc.asmx.....	53
Figure 3.12 InventoryDataSvc.asmx.....	54
Figure 3.13 ComponentControl.aspx.....	54
Figure 4.1 Number of incoming queries vs. Average Response Time with all HHs.....	57
Figure 4.2 Number of Web services vs. Average Response Time.....	59
Figure 4.3 Number of HHs vs. Average Response Time.....	60

ABSTRACT

Berbeco, Robert W., M.S., Purdue University, June, 2003. “The UniFrame .NET Web Service Discovery Service”. Major Professor: Rajeev Raje

Microsoft .NET allows the creation of distributed systems in a seamless manner. Within .NET small, discrete applications, referred to as Web services, are utilized to connect to each other or larger applications over a local or wide area network connection through HTTP. The Web services are written in Extensible Markup Language (XML) and registered with Internet Information Server (IIS), and can be applied in numerous fashions. This project uses the .NET capabilities to create a distributed discovery service (called as UNWSDS) that is an integral part of the UniFrame Approach. The UniFrame Approach (UA) provides a comprehensive framework which incorporates a meta-component model; a resource discovery service, called the UniFrame Resource Discovery Service (URDS); and generative programming and Quality of Service (QoS) to allow seamless interoperation of heterogeneous distributed components. The proposed UNWSDS incorporates the extensibility of Microsoft .NET through XML web services and distributed MS SQL 2000 servers into the URDS. A prototype is designed and implemented to validate the proposed UNWSDS. The results of this approach enable extending of UniFrame to incorporate .NET as another component model in it.

INTRODUCTION

As the Internet becomes more useful in today's world, users are, in turn, becoming reliant on Web-based services to fulfill our daily tasks. Along with the increased use of Web-based services comes the difficulty of locating the services and coordinating their usage; because in real Internet Web conditions, services are undiscovered, new and coming or going rapidly.

The ability to coordinate the usage of randomly distributed Web services across the Internet does not exist 1) since there are no set specifications in place for integrating all the information or locations of the individual Web services, and 2) because of the complexity required to develop the code to integrate these services [2].

Web services represent a paradigm shift from software in shrink-wrapped boxes to components or services that interested users could consume over the Internet. The Web services would be distributed across the Internet and interested users would locate and consume these services if they wanted to. The Web services could be available free, through subscription, or on a pay-per-use basis. In the past couple of years the interest in and utilization of Web services has grown considerably, leading to the introduction of development packages that take advantage of Web service capabilities. Some of these development packages include: 1) the Sun Microsystems, Inc. J2EE version 1.4, which includes full support for Web services, and the Java Web Services Developer Pack (Java WSDP) version 1.2, provides Web services capabilities and Java Web services APIs; 2) the SCO Group, Inc. SCOx which targets developers on the SCO Unix and Linux platforms; 3) BEA Systems's WebLogic Platform 8.1 which adds Web services capabilities to the BEA WebLogic Server 8.1; and 4) Microsoft's .NET Extensible Markup Language (XML) Web services which provides a framework for Web services on Microsoft OS systems. The large market share dominance that the Microsoft OS has on client workstations in the workplace and at home makes the use of .NET attractive for companies and individuals looking to quickly extend their Web services to the Internet or their existing infrastructures.

With the utilization of the .NET Extensible Markup Language (XML) Web services comes a need for an efficient search and retrieval mechanism, as the user will not typically know where a service is located. The URDS architecture provides services for automated discovery and selection of components which meet specific Quality of Service (QoS) requirements [4]. Not only would just .NET XML Web services be locatable within the URDS architecture, but also other components that are not developed specifically for Microsoft-based systems could be found. The incorporation of .NET within the UniFrame resource discovery framework would result in a robust, scalable, and transparent method for users to locate and consume .NET XML Web services of their choosing.

1.1 Motivation

In today's application development a shift is occurring from the desktop or server based components to thinner, Web service based clients and components accessible from the Internet. A Web service is truly a new breed of Web application. A Web service is a self-contained, self-describing, modular application that can be published, located, and invoked across the Web in a distributed fashion with the invocation and utilization being mostly transparent to the user [2]. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service. For applications or other services to use a Web service a dynamic discovery process that is efficient and secure must be employed. Currently there are several directory-based discovery methods that exist which are based on a publisher-subscriber model – wherein a service is registered or published to the directory and other services can subscribe or consume the service. Examples of some commonly used directory-based discovery methods include: Lightweight Directory Access Protocol (LDAP) which is an Internet protocol that email programs use to look up contact information from a server; Domain Name System (DNS) which is a global network of servers that translate host names such as `www.myserver.com` into Internet Protocol (IP) addresses, like `159.1.2.1`; Service Location Protocol (SLP), which is a Novell specific protocol that allows clients to locate

Novell Netware servers and other services on the network; and Universal Description, Discovery, and Integration (UDDI) which is specifically designed for publishing and locating Web services [5]. UDDI is based on existing Internet standards, primarily HTTP and XML, and is platform and implementation neutral. UDDI lacks dynamic discovery. Instead, it is extremely similar to a telephone book's white or yellow pages. Businesses list their services within the directory and clients browse or search the directory for the services they need. Although the registration process by a publisher can be done programmatically, the discovery process by a subscriber is not automatic. Either a user intervention is required or programmers must manually add the Web reference within their programs.

The motivation for the UNWSDS is as follows: as software begins to shift towards services instead of packaged programs, there is a need to automate discovery of these services in order to utilize them. As these services will be distributed randomly across the Internet in various locations, an infrastructure must be constructed to provide the capabilities to find and utilize the various available services. For instance, in the case of .NET XML Web services they can be distributed on numerous Internet Information Servers (IIS) throughout the world. To be able to utilize these services, a user must be able to find them and have methods for retrieval and consumption even though the user may not know where the service is located. With the integration of the concepts and functionality of Microsoft's .NET and compact frameworks, the URDS architecture would fulfill this need as it is designed to provide the infrastructure necessary to support universal dynamic resource discovery.

1.2 Objectives

This project entails the development of the UniFrame .NET Web Service Discovery Service (UNWSDS) in which .NET Web services could be located on desktops, laptops, and embedded devices either on a wired or wireless network.

The specific objectives of this project include:

- To propose the UNWSDS to take the dynamic discovery concepts of the URDS and implement them for the .NET Web services in Visual Basic.NET.
- To propose methods of discovering and consuming .NET Web services within the UNWSDS and to provide a user friendly interface for these methods.
- Develop a prototype for the UNWSDS and validate the principles behind the UNWSDS within the .NET and compact frameworks.

1.3 Contributions

The UNWSDS enables .NET XML Web services to be dynamically discovered utilizing the UniFrame Approach.

The contributions of this project include:

- The creation of an architecture, the UNWSDS which is based upon the URDS prototype [4], for the dynamic discovery service utilizing Microsoft's .NET and compact frameworks.
- Development of a UNWSDS prototype, wherein .NET XML Web services and embedded clients are dynamically discovered, written in Visual Basic.NET. The UNWSDS prototype is the expansion of the original URDS prototype [4] for the .NET platform.
- Validation of the above mentioned objectives by experimental testing.

1.4 Organization of this report

This project is organized into 5 chapters. The introduction of the motivation, objectives, and the contributions of the project were presented in this chapter. Chapter 2 surveys related work. Chapter 3 provides an overview, design details, and implementation of the project. Chapter 4 describes the prototype's validation by experimentation. Chapter 5 concludes this project with a discussion of what was accomplished and future work.

2. BACKGROUND AND RELATED WORK

This chapter provides the background and the related work that have contributed to the development of the UNWSDS. The related work is divided into three categories: 1) Microsoft .NET, 2) UniFrame, and 3) Discovery services. Section 2.1 discusses Microsoft .NET which was the platform used to develop the UNWSDS. Section 2.2 describes the UniFrame Approach in which the UNWSDS was modeled for. Section 2.3 describes various discovery services that exist, and how they compare and contrast to the UNWSDS.

2.1 Microsoft .NET

Microsoft .NET is a framework for developing web-based applications called Web services. The .NET framework is a three-tiered architecture for developing web-based applications: 1) the first tier is the Presentation or client-side interaction; 2) the middle tier is the Business logic tier; 3) and the remaining tier is the Data tier [15]. Figure 2.1 below represents the .NET three tiered system.

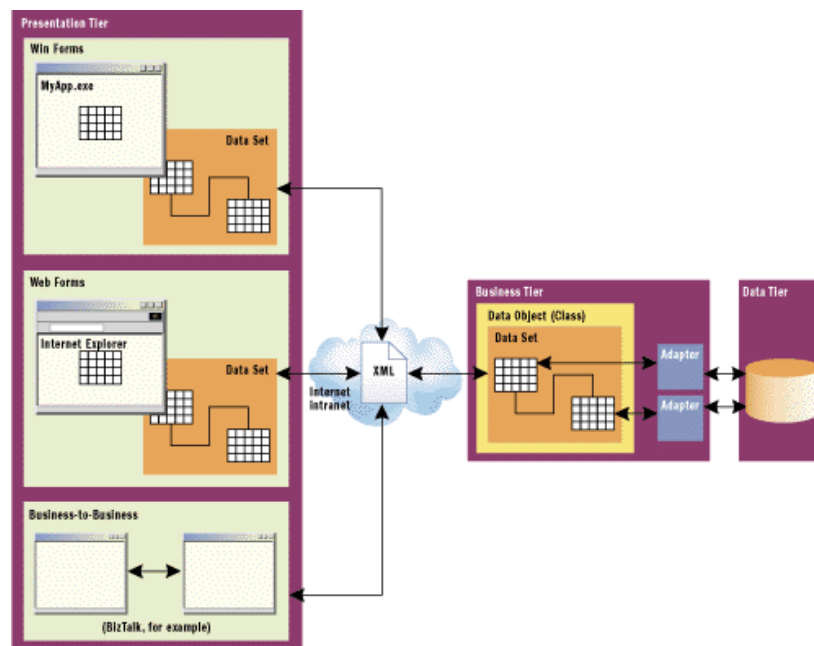


Figure 2.1 .NET three-tier system (from [15]).

Conceptually the .NET three tiered system is similar to Enterprise Java Beans (EJB). In the EJB three tiered system there are EJBs which are Java programs that complete the data processing of the middle tier and then communicate the results to the Java Server Pages (JSP) that present the data to the client. Whereas EJBs can only be written in Java, .NET programs can be written in multiple languages. In .NET, the programs which complete the data processing of the middle tier are Web services which can be written in C# or Visual Basic.NET. These Web services would reside on an Internet Information Server (IIS), which serves Web pages or Active Server Pages (ASP) and services to clients connecting via HTTP. IIS would fulfill the needs of the Presentation and Business tiers within the Microsoft .NET three tier model, and the Data tier could be fulfilled by any number of databases which could include Oracle or MS SQL.

The UNWSDS is developed using the .NET model with IIS and Web services fulfilling the role of the Business tier, and ASP.NET is used for the Presentation tier. Section 2.1.1 describes IIS and its security in detail. Section 2.1.2 describes Web services and their components. Section 2.1.3 describes ASP.NET.

2.1.1 Internet Information Server (IIS)

Internet Information Server (IIS) is Microsoft's answer to a Web server. IIS is a file and web application server that can be used on a local area network (LAN), a wide area network (WAN), or the Internet. IIS is usually installed as an additional component to a Microsoft OS server. The main services of IIS are: 1) World Wide Web (WWW), 2) File Transfer Protocol (FTP), 3) Simple Mail Transfer Protocol (SMTP), and 4) Network News Transfer Protocol (NNTP). Through the WWW service, IIS supports dynamic and static Web pages, and supports external data source connections to databases such as Oracle, MS SQL, or Access. The dynamic Web page content is developed using Active Server Page (ASP).NET technology, which means that applications such as Web services can be embedded in Web pages to present result content to clients. IIS is managed through the Microsoft Management Console (MMC), which is the default management

tool on a Microsoft OS server or workstation and interfaces with the Microsoft Windows 2000/XP server security model for permissions that also support Anonymous, Domain level, or server (local) level user authentication [16].

2.1.1.1 IIS Security

IIS has various security features to secure access to Web services. As IIS security is interfaced with Microsoft Windows 2000/XP, it is also integrated with the Kerberos security infrastructure. Users can be authenticated securely through Kerberos to IIS, and gain access to Web services or folders that they have permission to view or execute. In addition to client authentication, IIS can be configured for secure communications between server and client with Security Sockets Layer (SSL) and Transport Layer Security (TLS). Through SSL and TLS communication would be encrypted at 128-bit encryption and the server would verify the client before connection is granted. If further restrictions are necessary, Internet Protocol addresses and domains could be restricted. This would restrict certain or a block of computers from connecting to the server. In this way if IIS was necessary only for a company intranet, but did not need to be accessible from the Internet the company addresses could be added for access and the remainder denied.

A very important tool that Microsoft created and is downloadable from their web site is the IIS Lockdown Tool (currently 2.1). The IIS Lockdown Tool works by turning off unnecessary features and to provide multiple layers of protection against potential attackers [17]. When executed, the tool has a wizard interface that simplifies configuration as the server or IIS administrator would select what their server's primary use is, static content or Web services, and then the tool would configure the appropriate permissions. Another method of increasing IIS security is to configure the Microsoft Auto-Update feature. This can be scheduled daily to download and install any critical patches that need to be installed to ward off potential vulnerabilities.

2.1.1.2 Virtual Directories

An IIS virtual directory is a name mapped to a local folder or network share through the server for external access, and as such the virtual directory name is used for access instead of the local directory name. The virtual directory would be created through MMC and would require the following information: 1) name or URL for the virtual directory, 2) path of the local folder, and 3) Web service access or execute permissions enforced by IIS. In essence, a virtual directory is the path that an Internet user or application would use to access a Web service on IIS.

Once IIS has been installed, IIS security configured properly using the IIS Lockdown Tool, and potential critical updates applied with Auto-Update a virtual directory can be created. Once the virtual directory has been created Web services can be added to the folder for access by clients or other Web services.

2.1.2 Web services

Web services are a model for building applications that can be implemented and used by any computer system over the Internet. Whereas component-based object models like Distributed Component Object Model (DCOM), Remote Method Invocation (RMI), and Internet Inter-ORB Protocol (IIOP) depend on an object model-specific protocol; Web services extend these models by communicating with Simple Object Access Protocol (SOAP) and XML [8]. By using SOAP and XML, Web services are to communicate independent of object model-specific models or what client system is being used.

Web services perform functions or subroutines that can be simple requests to complicated business processes [7]. Essentially Web services are applications that expose a Web-accessible API and serve as interoperable building blocks for constructing applications. The Web services platform establishes a set of standards that applications

must follow to achieve interoperability over the Web. The services themselves can be written in whatever language and any platform desired, as long as they can be viewed and accessed according to the Web services standards; but they would have to be accessible from a Web server such as IIS or Apache. Essentially the Web services standards consist of three main components: Extensible Markup Language (XML) Schema Definition (XSD), the type system; Simple Object Access Protocol (SOAP), the invocation mechanism; and Web Service Description Language (WSDL), the description mechanism. Figure 2.1 below shows an example of a client invoking a Web service.

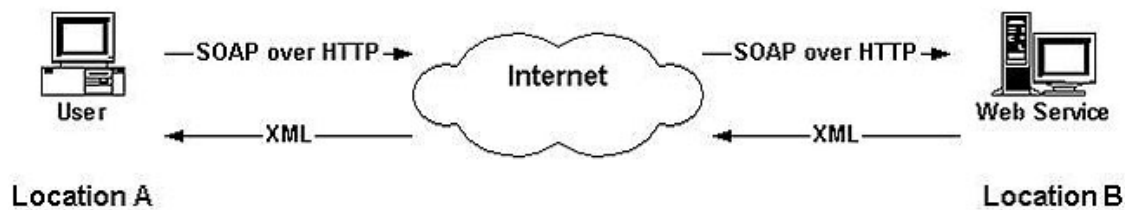


Figure 2.2 A client invoking a Web service via SOAP (from [8]).

XML is the format used for representing data through Web services. XML was chosen since it is platform and vendor independent, and relatively simple to parse. For the specification of built-in types and a language for defining additional types the XSD is used. When Web services are created in a specific programming language, the data types within have to be translated to XSD types to conform to the Web services standards. XSD can also be used to validate incoming messages within Web service communication to enforce the agreed-upon messaging contract between services.

When a Web service has been developed a mechanism has to exist for individuals or programs to invoke the service. SOAP provides this mechanism. SOAP is an application-level protocol and can interact directly over a transport protocol such as TCP and is layered over HTTP so that the communication can flow over the current Internet infrastructure. The layering of SOAP involves a SOAP message sent as part of the HTTP request or response. SOAP uses XML to achieve platform independence and interoperability when messages are exchanged between a client and a Web service.

Through the use of HTTP and XML, SOAP enables application-to-application communications regardless of platform or system infrastructure. XML-based messaging is the essence of SOAP and includes RPC which enables services to invoke other remote services via the Web. Through the leveraging of SOAP, client-side and server-side parts exist that handle serializing and deserializing application data into the appropriate XML format to send and receive the RPC messages.

The WSDL is an XML-based grammar for describing Web services, their functions, parameters, and return values [6]. Many development tools such as Visual Studio.NET can generate a WSDL document describing the developed Web service, consume a WSDL document, and generate the necessary code to invoke the Web service. In WSDL, a Web service exposes a group of methods or functions which are referred to as portTypes or interfaces. In order to invoke a method, a client sends an input message and gets the output message back. The input message would contain the data being sent to the service and the output message would contain the data being sent back from the Web service. Each item in the data of the message is called a message part and the protocol used to invoke the operation is specified in the binding. The service itself would be exposed to the world via one or many ports. Each port would represent the network address where the service is located and the binding required for the port. To summarize the use of WSDL in a top down hierarchy: 1) a Web service could contain one or more ports, and each port would reference a binding; 2) a binding references a portType, operations within the portType, and the messages that make up each individual operation with the portType; 3) a portType would contain zero or more operations; 4) an operation would have an input and output message; 5) a message would have zero or more parts and each part is of some data type; 6) and a part's type can be an XSD built-in type or custom defined using XSD [6].

2.1.3 ASP.NET

The UNWSDS is based on Microsoft's .NET and compact frameworks utilizing XML Web services in ASP.NET and written in Visual Basic.NET. Figure 2.3 displays the ASP.NET and ASP architecture.

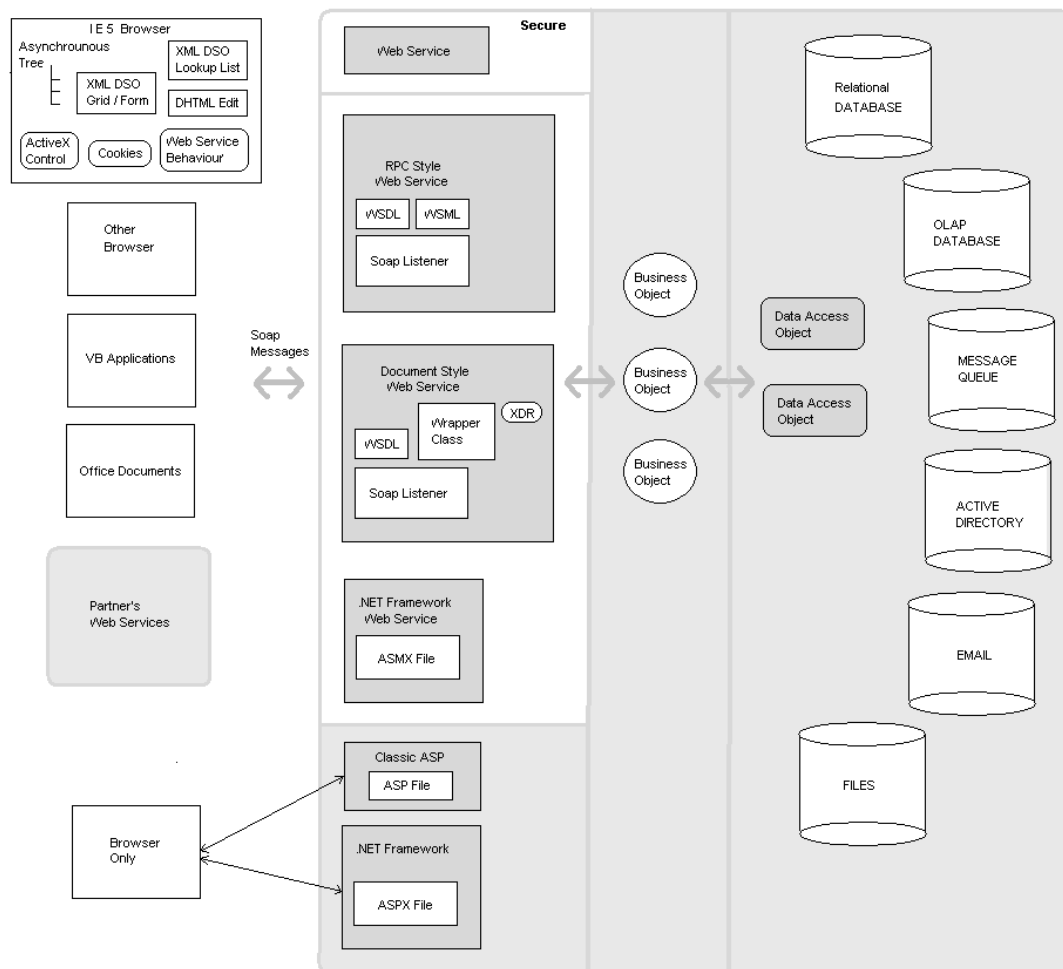


Figure 2.3 ASP and ASP.NET Architecture

The .NET framework has two main parts – the common language runtime (CLR) and .NET framework class library [13]. The common language runtime provides the common services for .NET applications and programs can be written in C, C++, C#, and Microsoft Visual Basic®, as well as some older languages such as Fortran. The .NET framework class library includes prepackaged sets of classes. Included within the library

are three main components: ASP.NET for building Web applications and services; Windows Forms for client user interfaces; and ADO.NET for connecting applications to databases. Figure 2.4 shows the components of the .NET framework.

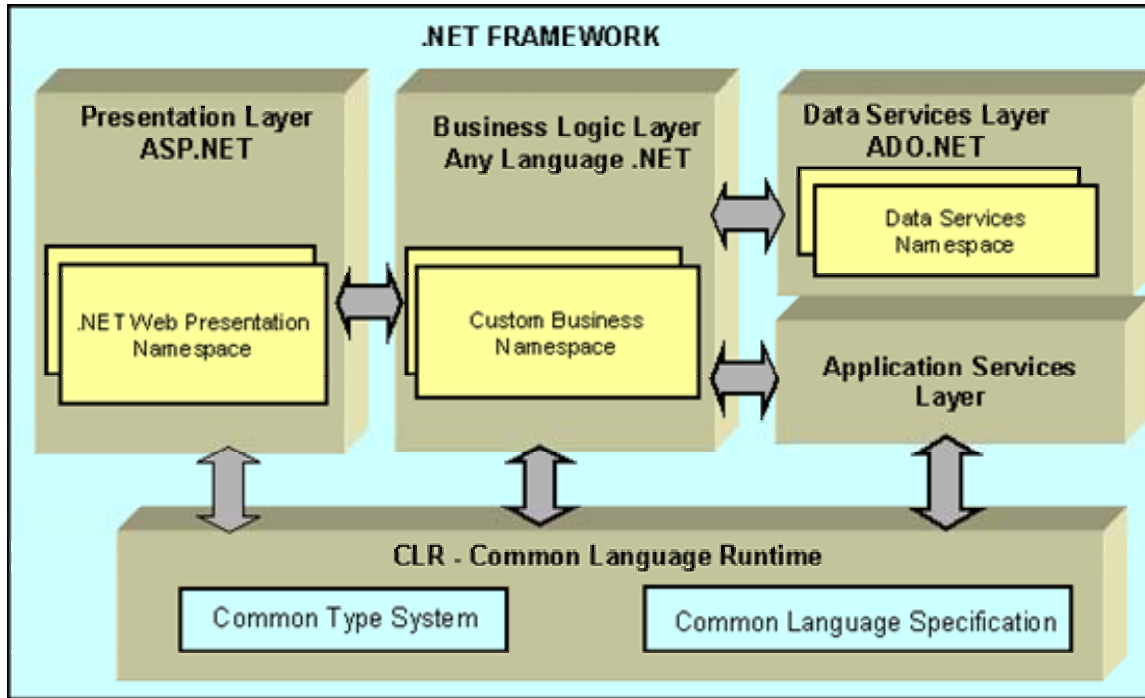


Figure 2.4 .NET Architecture (from [14])

ASP.NET enables a developer to use a full featured programming language such as C# (pronounced C-Sharp) or VB.NET to build web applications easily. All ASP.NET processes code on the Web server like a normal application and when the ASP.NET code has been processed, the Web server returns the results in HTML to the client. With the use of ASP.NET, Object Oriented Programming is available to build scalable and structured applications for the web. The traditional version of ASP uses HTML and VBScript to process or render pages, but due to the fact that VBScript is essentially a scripting language, developers are forced to write code with VBScript intermixed with HTML which was hard to read in large applications. ASP.NET separates the code from display in the HTML file as a separate file either in C# or VB.NET. In order to use the ASP.NET code, references can be added to the HTML pages so that events become controls and the ASP.NET code would execute the appropriate function or subroutine

based on user interaction. When writing the .NET XML Web services they would be written to utilize ASP.NET as ASP.NET has XML generation tools that make it fairly transparent to utilize XML for data storage, configuration, and data manipulation.

ASP.NET includes a large class library and encapsulates numerous common functions that can be used with an ASP.NET application. As an example from the project, data retrieval from a database was used to display the UNWSDS query results. In classic ASP this would have to be written manually with the development of the subroutines to handle the data connections and the data table display. In ASP.NET, the object DataGrid was used to bind result data from the Headhunters. The DataGrid would be rendered as a table on an aspx page and would display the bound data.

A key ability of ASP.NET is the ability to allow developers to write an application using multiple programming languages such as C#, VB.NET, or J#. One ASP.NET page would have to be written in a certain programming language, but each page could hypothetically be written in other languages and all the pages would be able to work together seamlessly.

2.2 UniFrame

The UniFrame Approach (UA) strives to provide a flexible and effective framework for developing and implementing distributed computing systems. The framework provided unifies distributed component models under the Unified Meta-Component Model (UMM) [3]. Within the UMM exist heterogeneous components, service and quality of service guarantees, and the infrastructure.

The components within UMM are autonomous and their implementation is non-uniform in the sense that there is no unified implementation framework, although they adhere to a distributed-component model. A UMM component consists of a state, an identity, a behavior, interfaces, private implementation, and three aspects which are computational, cooperative, and auxiliary. The computational aspect of a component is indicative of the task(s) completed, and depends on the task(s) objective, techniques

utilized to achieve the task(s) objectives, and specification of the component's functionality. The cooperative aspect deals with the interaction between components and contains expected collaborators (components that can potentially cooperate), pre-processing collaborators (component is dependent on other components), and post-processing collaborators (other components are dependent on this component). The auxiliary aspect handles additional features of a DCS, such as mobility, security, and fault tolerance.

Within UMM every component must specify the Quality of Service (QoS) specified for execution. The guarantee of QoS for each component is integral to any framework dependent on components being able to successfully complete the required tasks in a constantly changing heterogeneous environment. Within the UA, QoS of a DCS consists of a parameter catalog subdivided between static or design and dynamic or run-time parameters, formal specification of the parameters, and mechanism for ensuring the parameters [4].

The UniFrame Resource Discovery Service (URDS) provides the necessary infrastructure for UMM's discovery and communication mechanisms. Figure 2.5 displays the architecture of the URDS.

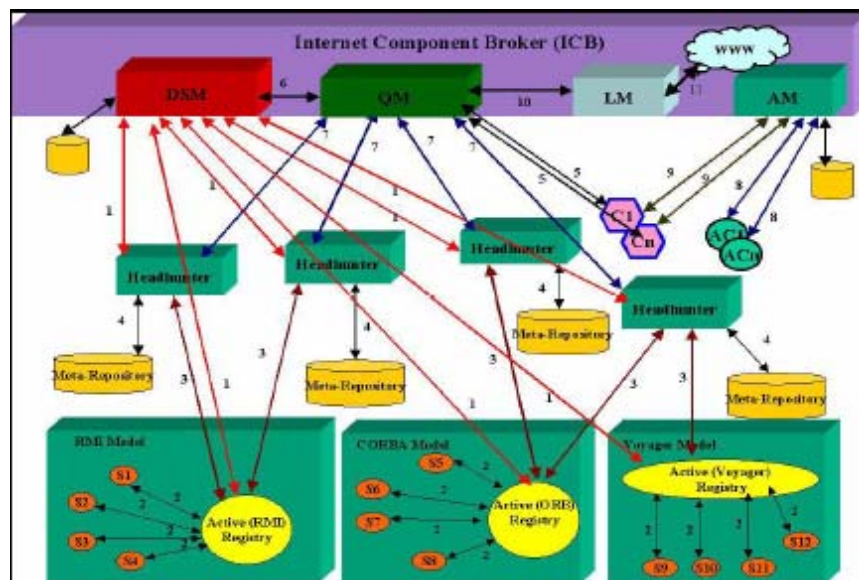


Figure 2.5 URDS Architecture (from [4])

The URDS consists of the Internet Component Broker (ICB) which in turn consists of the Query Manager (QM), Domain Security Manager (DSM), Link Manager (LM), and Adapter Manager (AM); Headhunters (HH); Meta-Repositories; Active-Registries; Services; and Adapter Components. The federated hierarchy of the URDS architecture promotes scalability and fault tolerance. Within the URDS every ICB is broken down into sub-components in the hierarchy represented by the Headhunters and each ICB is linked together with unidirectional links. Discovery in URDS is scoped by an administratively defined domain whereas each domain refers to an industry sector (i.e. Health Care, Manufacturing, etc.), and each domain is supported by the sector or organization providing the URDS service.

Finally, the URDS discovery process is based on periodic multicast announcements, and access control to multicast address resources and data encryption are utilized for data transmission security. More extensive security features for URDS data communication are planned for the future.

2.3 Discovery

Components can be stored and running randomly in a heterogeneous environment that can consist of embedded devices, laptops, desktops, and servers on a wired or wireless Local Area Network (LAN)/Wide Area Network (WAN). In order for a client to use one of these components a protocol has to be utilized in order for discovery. Discovery can be dynamic or directory based. Directory based methods of discovery include Universal Description, Discovery, and Integration (UDDI), Web Services Inspection Language (WSIL), Discovery of Web Services (DISCO), CORBA Trader Service [4], Lightweight Directory Access Protocol (LDAP) [4], Global Name Service (GNS) and Domain Name Service (DNS) [4]. Dynamic based methods of discovery include Service Location Protocol [4], JINI [4], Ninja Project: Secure Service Discovery Service (SSDS) [4], Salutation [4], Bluetooth [4], Universal Plug and Play (UPnP) [4], and Simple Service Discovery Protocol (SSDP) [4]. Since the various dynamic based

methods of discovery are covered in great detail in Nanditha Siram's Thesis [4], and with the primary focus of UNWSDS being .NET and XML Web services, they are not covered in the below subsections. UDDI, WSIL, and DISCO apply within the context of the UNWSDS, as they can be used for .NET and XML Web services, and are described further.

2.3.1 Universal Description, Discovery, and Integration (UDDI) Service

UDDI is a joint project among industry and business leaders initiated by Ariba, IBM, and Microsoft to encourage interoperability and adoption of Web services. The UDDI specifications provide a method to publish and discover Web services through a Web interface or programmatically from the UDDI Business Registry. The UDDI Business Registry is a logically centralized, physically distributed service that represents the core of UDDI and a XML file is used to describe a business entity and its Web services. A UDDI business registration consists of address, contact, and known identifiers; categorizations based on standard taxonomies; and technical information about services provided by the business which include references to specifications for Web services and URL based discovery mechanisms.

UDDI uses HTTP, XML, and SOAP to provide a uniform service description format and service discovery protocol. Web services are individually registered using UDDI discovery services and the information is added to the UDDI business registry either by a UDDI registration Web site or by using tools that make use of the programmatic service interfaces provided by the UDDI API. Once a Web service is registered with the business registry the data is replicated to all the UDDI root nodes, and soon becomes available to any person who utilizes the UDDI registry to discover the service.

The core information model used by UDDI is defined in an XML schema and represents four types of information used: business; service; binding; and specifications for the services. The UDDI businessEntity element represents Business information and

includes the XML elements for supporting publishing and discovering information about a business [9]. Figure 2.6 displays the businessEntity structure.

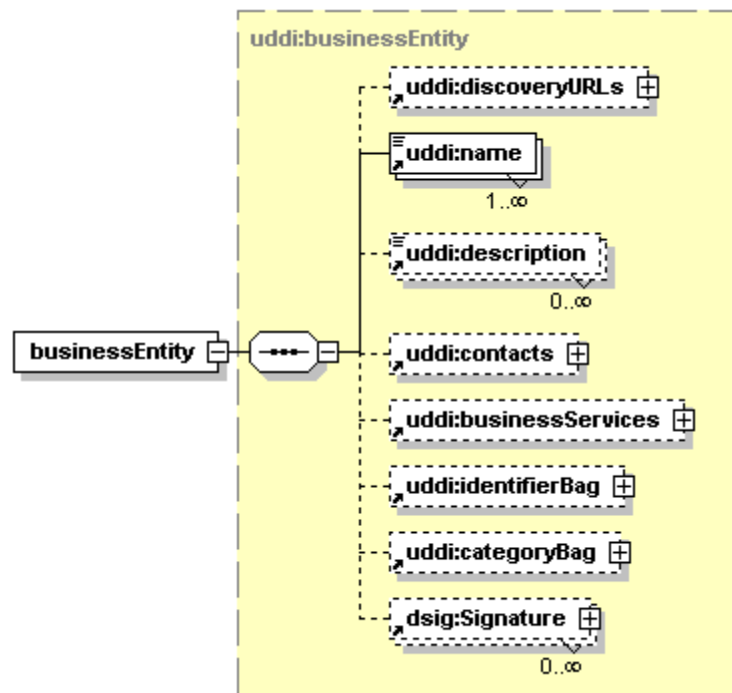


Figure 2.6 businessEntity structure (from [9])

The structure serves as a top-level information manager for all of the information about the business unit, and includes support for taxonomies that allow searches for a service in a particular industry, product category, or geographical region. A business registration consists of white, yellow, and green page information. The white page information consists of general information about the business such as business name, text description, contact information, and any other specified identifiers. The yellow pages function as a categorization of the business whereas the business is categorized based on standardized taxonomies (industry codes, product/services). The green pages specify how to bind to a service provider, and include technical information on how to invoke a business's services. For binding information, the bindingTemplate element is used to specify the necessary information about a published Web service to facilitate the invocation of that service. A set of references called tModels contain the meta-data about a specification, including the name, publishing organization, and URL pointer to the specifications themselves.

UDDI addresses quality of service issues by defining a calling convention involving cached bindingTemplate information in a retry on failure approach [9]. When a program is prepared for usage of a Web service, the bindingTemplate data is cached for use at run-time, then the cached bindingTemplate data is used when calling a remote Web service. If the call fails, a fresh copy of the bindingTemplate for the Web service is obtained from the UDDI Web registry and a comparison is made between the old and new information. If there is a difference, the failed call is retried, and if it is successful will replace the cached information with the newly obtained information.

Finally, the UDDI security model only allows individuals to publish or change information within the UDDI business registry [9]. The distributed UDDI business registry maintains a unique list of authorize parties and tracks which businessEntity or tModel data was created by a particular individual. A change or deletion is allowed only if the change request was made by the same individual who created the original information. All data communication with the UDDI business registry is through a secure channel.

2.3.2 Web Services Inspection Language (WSIL)

WSIL provides a mechanism for a service requestor to discover and utilize a XML Web service and was created with the intent to enhance the usability of UDDI [10]. In concept, WSIL is very similar to UDDI with the primary difference being that the Web service query does not communicate directly with the centralized UDDI registry, but is sent directly to the service providers from the requestor. WSIL takes the UDDI centralized service publication model, decentralizes it and distributes the individual components such that each Web service provider can advertise its own Web services rather than relying solely on UDDI. Instead of the centralized, un-moderated registry that UDDI offers, WSIL provides a decentralized, moderated registry distributed over the Web. UDDI and WSIL can interact in that a service can be registered with the UDDI registry and WSIL used to discover the service by redirection.

WSIL defines an XML based language which allows publishers to advertise their services and the elements for the language are defined in a specified schema namespace [10]. Within a WSIL document exists a root entity called a service, which is the wrapper for the service advertisements used for discovery. The WSIL specification defines a set of conventions consisting of fixed name and linked WSIL documents, which allow Web service requestors to locate WSIL documents. The fixed name WSIL documents are placed in a common entry point for a Web site (usually the root level) so that are easily found from an external search mechanism. The linked WSIL documents allow service providers to organize their Web service listings in a hierarchical manner and a hierarchy is established by organizing the links within the document. A combination of the fixed name and linker WSIL documents can be used to provide a well organized hierarchical layout of Web services provided in a similar format to a Web Site Map.

While WSIL is similar to UDDI it is not intended to be a replacement or competition, but rather an enhancement and WSIL has build-in support for interoperating with UDDI [10]. A WSIL document can provide a reference to an entry in a UDDI business registry in which the reference element has a location attribute (specifies the location URL of the UDDI registry) and a child element which specifies the UDDI key for locating the service within the registry. If the Web service has been properly registered with the UDDI registry, then the WSIL can be used to discover and connect to the service.

2.3.3 Discovery of Web Services (DISCO)

DISCO preceded UDDI and is a proprietary technology for publishing and discovering Web services developed by Microsoft. While DISCO is similar to UDDI in concept, it is meant more for simple document-based lookups and does not provide the extensive lookup capacity of UDDI or a Web service repository. DISCO is utilized to define a document format for a Web service, thus making it possible to be discovered. DISCO also has a discovery mechanism to find other Web services, determine their

capabilities, and interact with them [11]. To publish a deployed Web service with DISCO for static discovery, a .disco XML file needs to be created and placed in the root folder of the Web server. Within the .disco XML file, there are two important elements: the contractRef, which has two attributes – ref and docRef – used to point to the Web service WSDL and documentation; and the discoveryRef links DISCO documents to one another. These two attributes are all that is needed within the .disco file for the Web service.

Once a DISCO document has been created for a given Web service, there are two methods of discovery: a command line utility disco.exe which generates an output file about any Web services discovered for a given URL; and Visual Studio.NET provides a user interface in which a Web reference can be added and the connection to the Web service can be used within a VS.NET program. Whereas a .disco file provides a static discovery mechanism, dynamic discovery is enabled through the use of a .vsdisco file. When VS.NET is utilized for Web service development and an ASP.NET Web project or service is created a .vsdisco file is automatically created and placed in the Web server root folder.

Although DISCO is simpler to use than UDDI and quicker to develop for, it is extremely limited in scope and functionality. DISCO does not attempt to sort Web service information into categories or hierarchies to enable more sophisticated queries, whereas UDDI can. Since DISCO is proprietary it is limited to Microsoft-OS based platforms and development is primarily completed using Microsoft development IDEs. DISCO was the initial attempt to provide a registry and discovery service for Web services so that developers could become more familiar with the process. The eventual assumption was that if a more robust registry and discovery mechanism was necessary, then UDDI was to be used.

The Discovery services described in this chapter are mostly designed for systems that have been developed and deployed in a pre-configured or confined manner [4]. When using these systems, the Web services have to be defined and referenced within the programs that use them. There is not a method for dynamic discovery and these systems

cannot take advantage of heterogeneity and a constantly changing environment like the UNWSDS. The UNWSDS has been designed to leverage dynamic discovery, where Web services can be added and used as needed.

Chapter 3 provides the architecture of the UNWSDS. The design details and implementation will be discussed.

3. ARCHITECTURE

This chapter provides the architecture of the UNWSDS including the design details and implementation. The architecture of the UNWSDS uses the UniFrame URDS as the model.

3.1 URDS Architecture Overview

The URDS architecture is organized as a federated hierarchy [4]. Every ICB has zero or more Headhunters attached to it and the ICBs are in turned linked to one another. The URDS discovery process locates services within an administratively defined logical domain, and domains are determined by the organizations providing the service [4]. URDS discovery is based on multicasting and is designed to handle failures through periodic multicasted announcements and information caching in repositories. A lack of communication beyond a designated timeframe for the component indicates a failure and the system state is reset. Table 3.1 provides a brief description of each URDS component. Figure 3.1 is an illustration of the URDS architecture and its components.

Internet Component Broker (ICB)	The ICB acts as an all-pervasive component broker in an interconnected environment. It encompasses the communication infrastructure necessary to identify and locate services, enforce domain security and handle mediation between heterogeneous components. The ICB is not a single component, but a collection of services comprising of the Query Manager (QM), the Domain Security Manager (DSM), Adapter Manager (AM), and the Link Manager (LM). These services are reachable at well-known addresses. It is envisioned that there will be a fixed number of ICBs deployed at well-known locations hosted by corporations or organizations supporting this initiative.
Domain Security	The DSM serves as an authorized third party that handles the secret key generation and distribution and enforces group memberships

Manager (DSM)	and access controls to multicast resources through authentication and use of access control lists (ACL). DSM has an associated repository (database) of valid users, passwords, multicast address resources and domains.
Query Manager (QM)	The purpose of the QM is to translate a system integrator's natural language-like query into a structured query language statement and dispatch this query to the 'appropriate' Headhunters, which return the 32 list of service provider components matching these search criteria expressed in the query. 'Appropriate' is determined by the domain of the query. Requests for service components belonging to a specific domain will be dispatched to Headhunters belonging to that domain. The QM, in conjunction with the LM, is also responsible for propagating the queries to other linked ICBs.
Link Manager (LM)	The LM serves to establish links with other ICBs for the purpose of federation and to propagate queries received from the QM to the linked ICBs. The LM is configured by an ICB administrator with the location information of LMs of other ICBs with which links are to be established.
Adapter Manager (AM)	The AM serves as a registry/lookup service for clients seeking adapter components. The adapter components register with the AM and while doing so they indicate their specialization, i.e., which component models they can bridge efficiently. Clients contact the AM to search for adapter components matching their needs.
Headhunter (HH)	The Headhunters perform the following tasks: a) Service Discovery: detect the presence of service providers (Exporters), b) register the functionality of these service providers, and c) return a list of service providers to the ICB that matches the requirements of the component assemblers/system integrators requests forwarded by the QM. The service discovery process performs the search based on multicasting.
Meta-	The Meta-Repository is a data store that serves a Headhunter to

Repository (MR)	hold the UniFrame specification information of exporters adhering to different models. The repository is implemented as a standard relational database.
S1..Sn	Services implemented in different component models (RMI, CORBA, etc.,) identified by the service type name and the component's informal UniFrame specification which is an XML specification outlining the computational, functional, cooperational and auxiliary attributes of the component and zero or more QoS metrics for the component.
AC1..ACn	Adapter components, which serve as bridges between components implemented in diverse models.
C1..Cn	Component Assemblers, System Integrators, System Developers searching for services matching certain functional and non-functional requirements.

Table 3.1 Description of URDS Components (from [4])

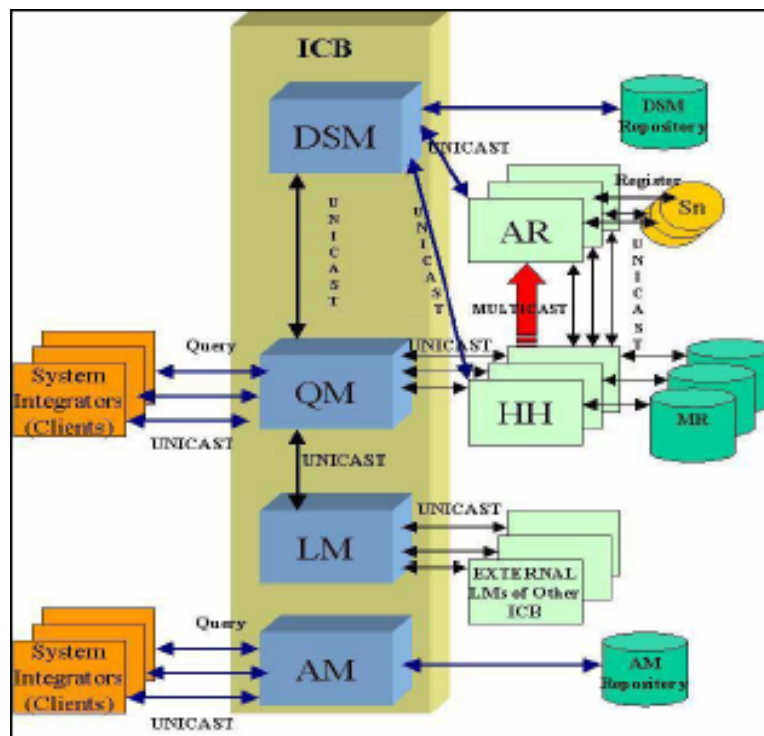


Figure 3.1 URDS architecture (from [4])

3.2 Design Details

The following subsections provide design details and algorithms for each of the URDS components that were designed, prototyped, and utilized within the UNWSDS using Visual Basic.NET. The following URDS components were utilized within the UNWSDS design and prototype: 1) Query Manager (QM), 2) Headhunters (HH), 3) Meta-Repository (MR), and 4) Active Registry (AR). These URDS components were chosen to be implemented in the UNWSDS as they can best exemplify the interaction of .NET XML Web services within UniFrame. Other URDS components were not developed within this timeframe, but would be added in the future enhancement set.

3.2.1 Query Manager (QM)

The QM is responsible for finding the Web services matching a user's requests. The QM parses the user's request into a structured query language statement and sends this query to the appropriate Headhunters (determined by the domain). Requests for Web services are sent to Headhunters and the Headhunters return the list of matching services to the QM. The QM obtains the list of Headhunters from the Headhunter Web service which in turn registers new Headhunters as Web services as necessary. Since the Headhunter Web service is registered with an IIS Web server, permissions can be configured to prevent or allow access on a domain level to serve the purposes of the DSM.

Selection of results by the QM is controlled by the user's parameter values in the query form. The user would be required to enter: 1) the Web service details such as domain, name, description, and function; 2) the functional attributes such as algorithms, complexity, and technology (in this case ASP.NET is the default); 3) search by Auxiliary attributes such as mobile or not mobile (being embedded device); 4) search by QoS Metrics such as end-to-end delay and availability. Each entered parameter would be considered as a constraint to the query. As more parameters are entered, the query is

constrained more, whereas a more global query would have very few parameter values entered.

The QM queries are handled in the following manner:

- Parse the user's entered parameters and extract the text pertaining to the various UniFrame specified attributes necessary for finding the Web services.
- Compose the extracted information into a SQL based query statement.
- Dispatch the SQL-based query to the closest (by network proximity) system's Headhunters.
- The Headhunters will query their associated databases and determine whether components exist that match the user's parameter values. If there is a match, the results are returned to the QM, if no match then the query is passed to a randomly selected active Headhunter. This process continues until the components are found or all the Headhunters within the domain have been queried. There is a time out, however, that is mentioned below. Although not currently implemented, a future enhancement for this process would include the ability to search Headhunters by network proximity based on the cost of communication and how quickly a Headhunter can respond to a query.
- The QM will wait for a specified time period for results to be returned from the Headhunters before timing out. Since the QM is implemented as a Web service in the UNWSDS, one of the connection options through IIS is `AspSessionTimeout`. The `AspSessionTimeout` call option is the length of period in seconds during which the Web service session will continue. If the Web service being queried does not answer, the calling Web service or QM in this case, will quit calling after this limit is surpassed. The `AspSessionTimeout` of 20 seconds was used for the QM. 1200 seconds is the default set by IIS, which would have dramatically increased the wait time of the client, so 20 seconds was set arbitrarily for this purposes of this implementation.

- The QM returns the results, whether there are results or not, to the user of the system. The results can be directly accessed by their Web service binding information displayed in the result table.

Data structures used within the QM:

Object queryObject	The query object holds all the attributes of the user's request which were selected as parameters for searching. The parameters include the Web service details such as domain, name, description, and function; the functional attributes such as algorithms, complexity, and technology; search by Auxiliary attributes such as mobile or not (mobile being embedded device); search by QoS Metrics such as end-to-end delay and availability. The query object also stores the necessary connection and authentication routines for communicating with the Headhunters via QM.
Object dataGridObject	Stores and displays the results by Web service ID and binding information.
Object sp_search_for_components	Database implementation of the search routine. This object actually executes the query based upon the queryObject's parameters. The primary purpose of this object is to increase the response time of the QM.

Table 3.2 Data structures used for QM functions

The QM and, subsequently, its sub-services are activated once it is configured as an IIS Web service application. As a Web service application, the QM stands ready to service incoming requests so no further initialization is required. IIS was utilized for restricting use of the QM within a local subnet by IP addresses. Only Web services or clients from the local subnet would be able to access the QM. To increase QM security, Web services or clients invoking the QM were not allowed anonymous access to IIS, but were authenticated securely by using specific pre-configured Web service accounts though Kerberos. The QM activation algorithm is displayed below. It is assumed that an IIS Web server has been pre-configured and is executing, and the QM Web service has been compiled.

```
QM_WEBSERVICE_ACTIVATION
    CREATE IIS_virtual_directory
    SAVE QM_Web_service within IIS_virtual_directory
    CONFIGURE IIS_virtual_directory as an application within
    the IIS MMC.
END QM_WEBSERVICE_ACTIVATION
```

Once the QM Web service has been activated it can process incoming requests. When a request has been received by the QM Web service it will parse the user's entered parameters and extract the text pertaining to the various UniFrame specified attributes necessary for finding the Web services. The UniFrame attributes used included Web service domain, name, description, function names, algorithms, complexity, mobility, end-to-end delay and availability. Next a SQL query will be generated based on the extracted information, and then the query will be sent to the closest (by network proximity) system's Headhunter Web services. The Headhunters will query their associated databases and determine whether components exist that match the user's parameters. If there is a match, the results are returned to the QM, if no match then the query is passed to to the next Headhunters. The Headhunters are chosen randomly for this implementation and this process continues until the components are found or all the Headhunters within the domain have been queried. The QM will wait for 20 seconds, which was set arbitrarily for this implementation, for results to be returned from the

Headhunters before timing out. The QM returns whatever results it receives to the user of the system. Once the results are displayed they can be directly accessed by their Web service binding information displayed in the result table.

The QM Client Query Handling algorithm outlines the process for servicing queries from clients. Client parameters are processed into a SQL query and then propagated to Headhunters. The results of the queries are returned to the client and each client is handled as a separate Web service session. The timeout period is pre-configured by the IIS administrator of the server running the QM, as 20 seconds. The following algorithm was adapted from Nanditha Siram's Thesis [4].

QM_CLIENT_QUERY_HANDLING

INPUT: *User_entered_parameters*

OUTPUT: *result_DataGrid*

/ The Headhunters will query their associated databases and determine whether components exist that match the user's parameters. */*

WHILE *incoming_request* = TRUE

Parse user_entered_parameters and generate SQL

Create a new SQL query entity

n = 0, timeout_count = 0

/ If there is a match the results are returned to the QM, if no match then the query is passed to next nearest (by network proximity) system's Headhunters. This process continues until the components are found or all the Headhunters within the domain have been queried. The QM will wait for specified time period for results to be returned from the Headhunters before timing out */*

 WHILE *request_results_found* = FALSE

 OR *timeout* = TRUE

Send SQL query entity to

HeadHunter[n]_Web_service

HeadHunter_Web_service[n] query meta-data


```

/* No results, will have to query next nearest
HeadHunter */
IF HeadHunter_Web_service[n] results = 0
    request_results_found = FALSE
    timeout_count++

// Results found
ELSE
    request_results_found = TRUE
END IF

// Bind results to DataGrid for output
Add results to result_DataGrid
n++
END WHILE

/* Web service form will output the results in
the DataGrid */
Send response to client with result_DataGrid
END WHILE
END QM_CLIENT_QUERY_HANDLING

```

User parameter values are entered through the main UniFrame query form and then the text is extracted for generating the appropriate SQL query. Once the SQL query is ready for processing it is passed to the Headhunters for searching. Since database stored procedures are much more efficient to utilize than generating SQL queries ad hoc, the Headhunter queries are pre-defined with the expected SQL parameters. If there are results these are bound to the DataGrid object. This algorithm was adapted from Nanditha Siram's Thesis [4], with the revisions being specific to Web service communication. This process is summarized below.

```

QM_PROCESS_USER_PARAMETERS
    INPUT: domain, componentName, componentDescription,
           functionNames, algorithms, complexity,
           technology, availabilityValue,
           end2endDelayValue, mobility as string

```

```

OUTPUT: result_DataGrid

/* All input is bound to parameters that are used by the
database stored procedure */
Bind input parameter values to stored procedure parameters

/* The database stored procedure is invoked to find the
requested Web services */
INVOKE dbo.sp_search_for_components

/* Results would be bound to DataGrid object for
output */
BIND result_DataGrid
END QM_PROCESS_USER_PARAMETERS

```

The DataGrid object allows for user or client sorting of the retrieved Web services. Essentially these are implemented as links within the DataGrid object itself. Clicking on the appropriate column header link would sort by that parameter. The advantage of binding the resulting information from the Headhunters with the DataGrid is another query is not necessary to sort the data. The sorting routine is embedded within the DataGrid itself. This actual process of sorting is summarized below.

```

QM_SORT_RESULTS_DATAGRID
  INPUT: DataGrid_sort_by_parameter
  OUTPUT: DataGrid_sort_results

  IF on_click DataGrid_heading_link = TRUE
    DataGrid_sort_by_parameter = on_click
    DataGrid_heading_link
    DataGrid_sort_results = descending_sort_by
    DataGrid_heading_link
    RETURN DataGrid_sort_results
  END IF
END QM_SORT_RESULTS_DATAGRID

```

3.2.2 Headhunters (HH)

The Headhunters within the UNWSDS are used for Web service discovery and passing the list of Web services and providers to the ICB based on the client entered parameters.

The Headhunters will query their associated databases and determine whether components exist that match the user's parameters. When a query is passed to the first Headhunter, the Headhunter queries its repository. If no results are found, the query is passed to the next randomly selected active Headhunter, and the process continues until results are found or the query times out. For this implementation the Headhunter selection process is random, but a future enhancement would include selection based on network proximity or time associated with communication. For the UNWSDS each result table for each Headhunter was displayed as a DataGrid so one could see how the query process was working.

Once the Headhunter services are loaded within the UNWSDS they stay active awaiting queries. On a periodic basis they refresh their meta-data repositories. This refresh time period is determined within the Web service itself and can be reconfigured by the administrator of the UNWSDS for the localized domain, such as a company or educational institution. For the purposes of this project, the refresh time was set arbitrarily at ten minutes. Too low a time set for refresh, such as 1 minute, would have a detrimental effect on the performance of the UNWSDS, whereas too high of a time would reduce the Headhunter's abilities to maintain a current list of accessible Web services. IIS virtual directory hierarchy was established so that the Headhunters could efficiently look for the available Web services. Any module or .asmx file (these are considered .NET web services) in the Web Services virtual directory on IIS used within the UNWSDS were assumed to be available services by the Headhunters. Web service enabling in .NET done once a Web service has been added to a virtual directory for Internet access it is assumed to be active. Web services can be "deactivated" by removing the Internet access to them via IIS. Essentially a Web service in .NET is "not active" for location by the Headhunters when the Headhunters cannot find them at the prescribed

virtual directory location either due to security access removal or removal of the service itself. The Headhunters can be programmed to locate Web services individually or can have the same Web services in their meta-data tables based on how the Headhunter Web service is configured.

Another nice feature of using .NET can be leveraged by using the Microsoft SQL 2000 servers for the Headhunter meta-data repositories. If necessary, each meta-data repository can be configured for merge replication amongst the individual servers. The database merge replication would have to be configured by the UNWSDS administrators within a localized domain, such as a company or educational facility. The UNWSDS administrators within a localized domain would be the logical choice for the configuration of replication since they will need to configure the meta-data repositories for the Headhunters. Adding the merge replication would involve a relatively minor change to the default database schemas. These SQL servers can be located on desktops, laptops, and embedded devices and can be connected via wired and wireless connections. What is advantageous about the merge replication is it can be configured for immediate data synching when the database is mostly online or periodical data synching when the database can be offline for extended periods like in the case of embedded devices. The advantage of this replication is the synching of current active Web service data being known to multiple Headhunters without the need to do this programmatically. The disadvantage is one would be tied to the proprietary Microsoft SQL 2000 server product to accomplish this goal. Unfortunately, this is in conflict with the goals of interoperability in the UniFrame Approach, and for the purposes of the UNWSDS replication was not explored with the Headhunters. However, for the sake of exploration, the Bank and Manufacturing databases used by the Financial and Manufacturing domain Web services did use merge replication.

Data structures used within the Headhunters:

Object componentTable	Mapping of the Web services directories and binding information necessary for clients to consume services. This object
-----------------------	--

	also stores the detailed UniFrame Specification information necessary for client queries.
--	---

Table 3.3 Data structures used for Headhunters

As the Headhunters are considered sub-services of the QM in the UNWSDS, they are activated once the QM is configured as an IIS Web service application. Since the Headhunters are Web service applications like the QM, they stand ready to service incoming requests and no further initialization is required. The Headhunters activation algorithm is displayed below. It is assumed that IIS has been pre-configured and executing, and the Headhunter Web services have been compiled.

```
HEADHUNTER_WEBSERVICES_ACTIVATION
```

```
  INSTALL HeadHunter_Web_services within QM_virtual_directory
  VERIFY QM_virtual_directory configured as application within
  MMC
```

```
END HEADHUNTER_WEBSERVICES_ACTIVATION
```

As mentioned earlier, IIS virtual directory hierarchy was established so that the Headhunters could efficiently look for the available Web services. Web services in the Web Services virtual directory on IIS used within the UNWSDS were assumed to be available services by the Headhunters. This process is summarized below.

```
POPULATE_HEADHUNTER_DATA_TABLES
```

```
  INPUT: directory_location
```

```
  /* Get the handle to the directory_location for the
  'active' Web services */
  web_service_location = directory_location
```

```
  /* Obtain the component data stored in this Web service
  directory */
```

```
  FOR EACH web_service IN web_service_location
    web_service_binding = LOOKUP web_service
```

```

/* Separate Web service binding parameters from the
Web service binding information */
web_service_binding_parameters =
LOOKUP web_service_binding

/* Obtain the Web service data stored in the binding
parameters */
componentTable = web_service_binding_parameters

/* Store the Web service data from the componentTable
into the HeadHunter meta-repository */
IF componentTable has elements
    component_information =
    LOOKUP componentTable_elements
    INSERT component_information IN
    HH_meta_repository
END IF
NEXT // End For
END POPULATE_HEADHUNTER_DATA_TABLES

```

A SQL stored procedure is used by the Headhunters to query their respective meta-data repositories or databases. A SQL stored procedure is a piece of code written for and stored on a database server that performs one or more operations. The code within a stored procedure can be a batch or statement that can be called from applications or Web services and can pass results back to the calling application. The use of a stored procedure is more efficient and has a quicker response time than an ad hoc query. For the Headhunter to be able to connect and use it, the stored procedure, will reside as callable code within the Headhunter's meta-data repository. The stored procedure process is displayed as SQL below.

```

Dbo.sp_search_for_components
    @domain as varchar(30),
    @componentName as varchar(100),
    @componentDescription varchar(1000),
    @functionNames varchar(500),

```

```

@algorithms varchar(200),
@complexity varchar(30),
@technology varchar(30),
@availabilityValue varchar(30),
@end2endDelayValue varchar(30),
@mobility varchar(5)
select '<a href="' + ID + '" target="new">' + ID + '</a>'
"BINDING", NAME, DESCRIPTION, PID, '<a href="' + CLIENT + '"
target="new">' + CLIENT + '</a>' "PROXY", END2ENDDelay,
AVAILABILITY, MOBILITY from COMPONENT
where DOMAIN LIKE @domain AND
NAME LIKE @componentName AND
DESCRIPTION LIKE @componentDescription AND
THE_FUNCTION LIKE @functionNames AND
ALGORITHM LIKE @algorithms AND
COMPLEXITY LIKE @complexity AND
TECHNOLOGY LIKE @technology AND
AVAILABILITY LIKE @availabilityValue AND
END2ENDDelay LIKE @end2endDelayValue AND
MOBILITY LIKE @mobility AND
ACTIVE = 'Y'

```

As mentioned previously, the Headhunters will refresh their meta-data tables every ten minutes to verify previous existing Web services are still active and consumable. If Web services have been removed from their Web service directory, then the Headhunters will remove them from their data tables. This refresh is similar to the process for populating Headhunter data tables, except refresh focuses on Web services that have been removed or could not be located and those that have been newly added. Whereas populating Headhunter data tables involves adding all the locatable Web services to the meta-data repositories, the refresh selectively removes those that cannot be located and adds those that have been newly added. This process is shown below in more detail.

```

REFRESH_HEADHUNTER_DATA
INPUT: directory_location

```

```

/* Get the handle to the directory_location for the
'active' Web services */
web_service_location = directory_location

/* Obtain the component data stored in this Web service
directory */
FOR EACH web_service in web_service_location
    IF web_service_exists_in_HH_data_table = FALSE
        web_service_binding = LOOKUP web_service

        /* Separate Web service binding parameters from
the Web service binding information */
        web_service_binding_parameters =
LOOKUP web_service_binding

        /* Obtain the Web service data stored in the
binding parameters */
        componentTable = web_service_binding_parameters

        /* Store the Web service data from the
componentTable into the HeadHunter meta-
repository */
        IF componentTable_has_elements = TRUE
            component_information =
LOOKUP componentTable_elements
            INSERT component_information INTO
HH_meta_repository
        END IF
    ELSE IF web_service_exists_in_HH_data_table = TRUE AND
LOOKUP web_service = FALSE
        DELETE component_information FROM
HH_meta_repository WHERE
web_service_binding_parameters = web_service
    END IF
NEXT // End For

```

If there are too many Headhunters within the UNWSDS, there could be degradation of performance to the overall system. Further analysis of this effect is found

in the chapter 4 experiments done with the UNWSDS prototype. The administrator of the UNWSDS for a localized domain would have the authority to remove Headhunters when necessary, such as for performance or testing purposes. The Headhunter, since it is implemented as a Web service, is removed from the system by being removed from its IIS application virtual directory. Web services can be removed from access in other ways (removing security being one), but this is by far the cleanest way of ensuring an application exception error does not occur on the client side. The process for removing a Headhunter Web service is displayed below. This process is a Web service specific adaptation from the Headhunter removal algorithm outlined in Nanditha Siram's Thesis [4].

```
REMOVE_HEADHUNTER
```

```

    // Close active processes
    IF open_communication = EXISTS
        DEACTIVATE Headhunter_open_communication in IIS
    END IF

    // Remove Headhunter Web service from its virtual directory
    REMOVE Headhunter
END REMOVE_HEADHUNTER
```

3.2.3 Meta-Repository (MR)

The Meta-Repository is a data store or database that holds service information of components adhering to different models [4]. The Meta-repositories within the UNWSDS are implemented as SQL 2000 database instances that are configured on multiple systems. Within the SQL 2000 database instances, there is UniFrame schema. The UniFrame schema serves to control access permissions, and the data store for the meta-repository information within the schema is the Component table. The MR organization is described in more detail within section 3.3.2.4.2. These databases, in turn, house the data tables and stored procedures used by the Headhunter for Web service information storage and retrieval. Within the Web service data table the service type name, binding

information, specification details, and QoS values are stored. As mentioned earlier, the SQL 2000 databases could be configured for merge replication to maintain the Headhunter data across each individual data table, but this would create a reliance on the Microsoft developed proprietary system and would go against the basic principles of the UniFrame Approach.

3.2.4 Active Registry (AR)

The active registry in the UNWSDS is the Microsoft Internet Information Server or IIS itself. IIS allows the creation of virtual directories and the ability to enable them as applications. A virtual directory in IIS is a system that allows administrators to put Web services into private directories on the Internet, and based on the established permissions, allow other Web services to connect to and communicate with the services placed into the virtual directory. The use of virtual directories is highly useful as it is relatively easy and quick to publish a Web service for external Internet access. By completing this process you are in effect “registering” the Web services with the IIS server and greatly simplifies the process of Active Registration. The virtual directory becomes the Web service(s) location and is recognizable by IIS once it is configured as an application, so a separate AR is not absolutely necessary.

3.3 Implementation

This section describes the UNWSDS prototype implementation and contains a description of the technology used and the prototype implementation itself.

3.3.1 Technology Used

This section describes the various systems used to implement the .NET prototype and how they interacted within the .NET URDS architecture.

Internet Information Services is a Windows-OS component that enables system administrators or system owners to configure their systems as Web servers. A Web server serves web pages to clients across the Internet or an Intranet. The Web server hosts the pages, scripts, services, applications, and multimedia files, and serves to clients using HTTP. A Web server is locatable by its IP address via HTTP, and in most cases the IP address is DNSed to facilitate easier location by a URL instead of the IP octet. In addition to IIS another very commonly used Web server is Apache. IIS 5.1 is used as the Web server for the UNWSDS.

A database server is software or service that manages a database or multiple instances of databases. A database server includes a Database Management System (DBMS) which is responsible for maintaining and managing the data in the database, the access privileges, transaction synchronization, and backup. The DBMS enables clients to manipulate and extract data from the data tables within the database. If a client needs to obtain information from the database, a request or query written in a “Structured Query Language” would be made to the database through the DBMS. Assuming the client has the proper access permissions in place, the subsequent results, if any, would be displayed. Data manipulation would be completed using a very similar method.

Typically, database servers can cope with many clients connecting at once without any problems. In the case of multiple clients wanting to update the database at once, the DBMS would synchronize the transactions without corrupting the data within the database. A database server usually contains one or more databases, in which the data is stored in an actual file accessible by the DBMS. Within the database there are one or more tables and many other objects (Triggers, Stored Procedures, Views) that would facilitate data manipulation. Within a table are one or more records also known as rows. Tables can be related to each other by primary (parent) and foreign (child) relationships. Microsoft’s SQL Server 2000 and CE 2000 are used within the UNWSDS for the prototype. SQL Server 2000 is the Windows-OS based software usable on desktops, servers, and laptops, while SQL Server CE 2000 is used on embedded devices.

An application server is a software package that serves as an intermediary between a web server and a back-end system, such as a database or legacy/mainframe application [12]. A Web client request is received by the Web server, which sends the request to the application server. The application server parses the request and communicates with the back-end systems to return the appropriate HTML response to the Web server for sending back to the requesting client. Some other examples of application servers are ASP.NET, Allaire Cold Fusion, PHP, and J2EE™. For the purposes of the UNWSDS, the application server used is ASP.NET.

3.3.2 Prototype Implementation

This section describes the implementation of the UNWSDS architecture. Figure 3.4 illustrates the implementation. The UNWSDS prototype is implemented on multiple systems; for the sake of experimentation, desktops, laptops, and embedded devices were used. The main issue in implementation of the UNWSDS occurred between the embedded devices and workstations. As embedded devices have much less storage capacity and memory available for application execution than workstations, these constraints served as a performance bottleneck for the implementation. Also due to the lack of system resources available to an embedded device, large complex applications would not be functionable. The largest hurdle for embedded UNWSDS was the lack of a mature IIS application to host the Web services required for full prototype functionality. Unfortunately, there is not an ASP.NET supported IIS version for embedded devices currently. In order to simulate Web services on the embedded device for the prototype, they had to be manually added to the embedded device by synching the Web service WSDL files between a workstation connected to the embedded device. The WSDL files could be found by the UNWSDS Headhunters through the same IIS methods available to a workstation, except they could not be dynamically invoked. A future enhancement to the UNWSDS would be to remedy this issue as soon as a more functionable IIS, that takes advantage of ASP.NET, is available to embedded devices. To keep in line with the goals of the UniFrame Approach the UNWSDS is a distributed, multi-tier architecture.

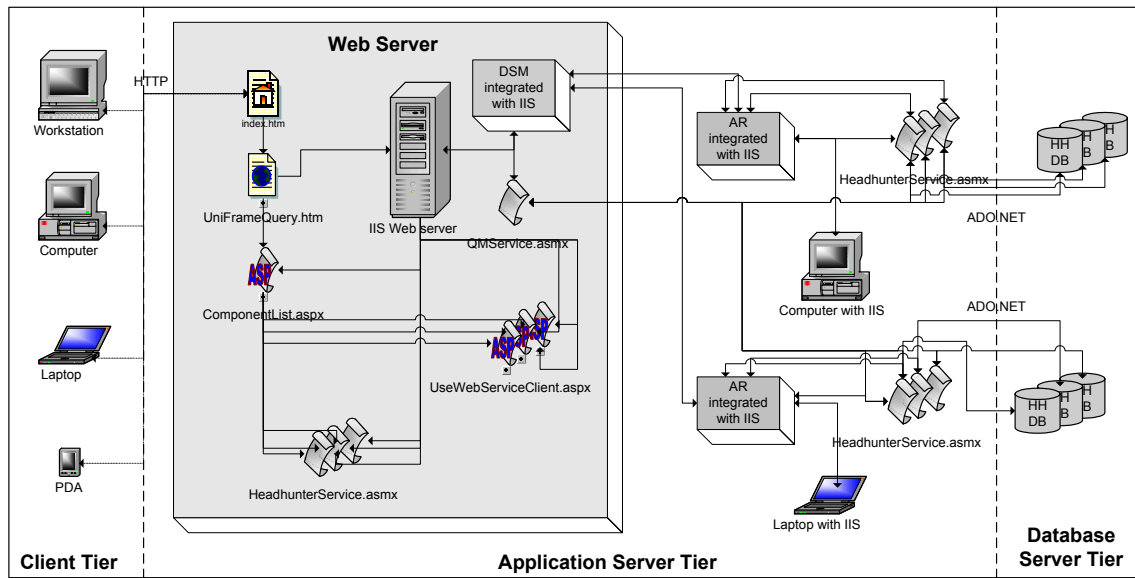


Figure 3.2 UNWSDS Implementation

In the UNWSDS, the client tier is configured to support a variety of client types through a web-based interface. The application server tier supports client services through the Web Server and also supports the various UNWSDS components (QM, HH, AR) as .NET XML Web services. The database server tier supports access to the HH database repositories by means of standard ADO.NET.

3.3.2.1 Environment

Microsoft .NET is used to implement the various services of the UNWSDS prototype. The .NET XML Web services (QM, HH, AR, domain service applications, etc.) are implemented using Visual Basic.NET within ASP.NET as asmx files. The data repositories (HH Meta-data, domain service applications) are implemented as databases on Microsoft SQL Server 2000, wherein Publisher and Subscriber Replication has been

enabled for the domain service applications. Scripted application web pages that service client queries are implemented in ASP.NET as aspx files.

3.3.2.2 Communication

Communication between the core components is achieved through the SOAP protocol between the various .NET Web services. The connections to the SQL 2000 databases are established using ADO.NET. Interactions between the clients and the other non-Web service components are based on the HTTP protocol.

3.3.2.3 Security

The security infrastructure in the UNWSDS is maintained through IIS user authentication either on the local system or domain level and can be enhanced by SSL communication. IIS user authentication is done through NTLM or Windows Integrated Authentication where the user's authentication parameters are verified with the local system or domain level group access tables. This, in turn, determines the access level and execution level of the services being accessed. In addition to this level of security IP filtering can be utilized to limit access to certain subnets or IP addresses.

The Secure Sockets Layer or SSL provide clients to server encryption for all data passed between the web server and client. In order to use SSL, a secure certificate would have to be purchased from a provider such as www.verisign.com. First, a certificate request must be generated, and Verisign will issue a certificate for installation and configuration. Once the certificate is installed, secure communication is established using https:// instead of http:// in the site URLs. All service communication from this point between server and client would be encrypted.

3.3.2.4 Programming

As mentioned earlier, .NET XML Web services were used in the implementation of the prototype, and to provide URDS functionality. Each service or database object represents specific behaviors, manipulation of data, or data storage in the application.

3.3.2.4.1 Service Objects

Individual service objects are used to represent the individual results of a database query or to exemplify a service in terms of an object. These service objects communicate with each other and external clients through SOAP and/or HTTP. The prototype contains the following service objects: ComponentControl, ComponentList, Components, Global, Headhunter, UniFrameIndex, UniFrameQuery for the UNWSDS. The prototype also contains the following service objects: BankDataSvc, InventoryDataSvc, GetCustomerAccountsClient, GetInventoryAccountsClient, MyProxyClass for the individual Web services that are used as Domain services by the Headhunters. The UNWSDS class diagrams are displayed in Figure 3.3 and the Domain service class diagrams are displayed in Figure 3.4.

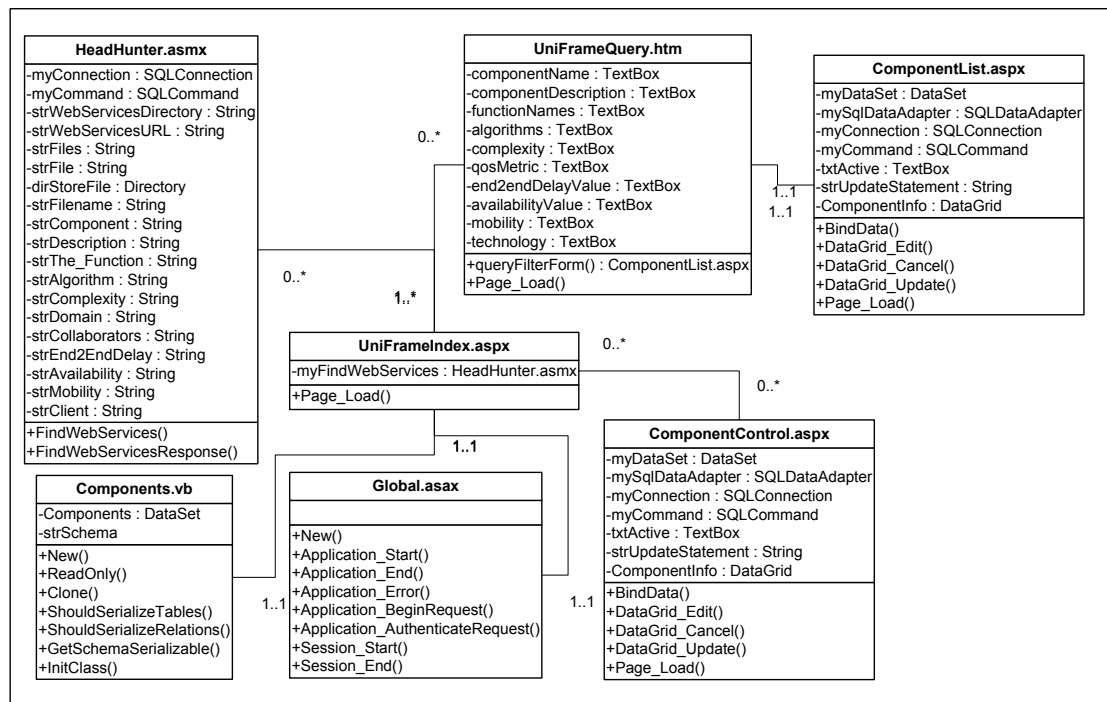


Figure 3.3 Class Diagrams for the UNWSDS Services

The following is an explanation of the above UNWSDS services.

- `HeadHunter.asmx` – this service represents the UniFrame UNWSDS Headhunter. The web service refreshes the Headhunter meta-data tables based on the following algorithm: 1) each Headhunter looks for active web service components; 2) each Headhunter finds active services and adds the service information, including binding, to its meta-data tables; 3) periodically the Headhunters would complete a meta-data refresh, using the first 2 steps again.
- `Components.vb` – this component is used to serialize XML data readable through SOAP for communication between the UniFrame services.
- `UniFrameQuery.htm` – this web page takes user input from the form and passes it to `ComponentList.aspx` for component searching via Headhunters.
- `UniFrameIndex.aspx` – this web form is the main point of entry for the prototype. It initializes the UniFrame UNWSDS and provides the portal for user interaction.
- `Global.asax` – this component contains the methods for responding to application-level events raised by the `HttpModules` in the UNWSDS services.
- `ComponentList.aspx` – this web form takes user input from `UniFrameQuery` form, processes it, and passes to Headhunters for searching. Results are displayed via `DataGrid`. Within the `DataGrid`, the user can select to consume the service directly through the service's client interface, or view the WSDL definition of the Web service.
- `ComponentControl.aspx` – this web form provides an administrative interface for the UniFrame UNWSDS for testing purposes. When invoked, it will display all

known currently accessible Domain Web services in the UNWSDS. Once the list is displayed, the administrator can enable or disable the Web service to the Headhunters that would be searching for the service.

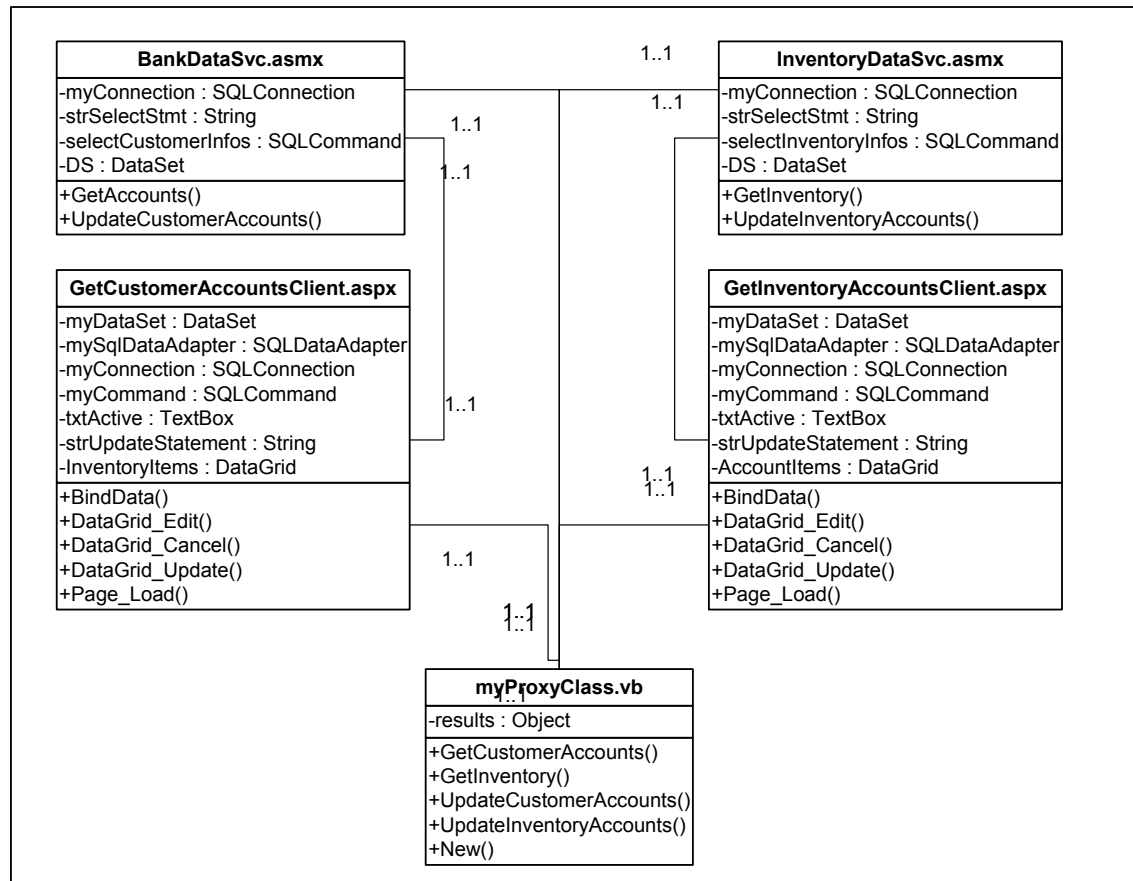


Figure 3.4 Class Diagrams for the Domain Services

The following is an explanation of the above Domain services.

- **BankDataSvc.asmx** – the Bank Data Web Service. Will query for bank account information and allow updates based on user input.
- **InventoryDataSvc.asmx** – the Inventory Data Web Service. Will query for inventory account information and allow updates based on user input.

- GetCustomerAccountsClient.aspx – this web page consumes the GetAccounts detail function in the BankDataSvc web service using the web service proxy myProxyClass.vb. Results are displayed and editable through the DataGrid.
- GetInventoryAccountsClient.aspx – this web page consumes the GetInventory detail function in the InventoryDataSvc web service using the web service proxy myProxyClass.vb. Results are displayed and editable through the DataGrid.
- myProxyClass.vb – this component is the proxy for the GetInventoryAccountsClient.aspx and the GetCustomerAccountsClient.aspx to directly access the InventoryDataSvc.asmx and BankDataSvc.asmx services respectively.

3.3.2.4.2 Database Objects

The database objects used in the prototype serve as data storage for the various UNWSDS and Domain level Web services. Database access is through user authentication within the Security component in MS SQL 2000 Server. Database merge replication has been enabled for the Manufacturing and Inventory schemas. The database objects class diagrams are displayed below in Figure 3.5.

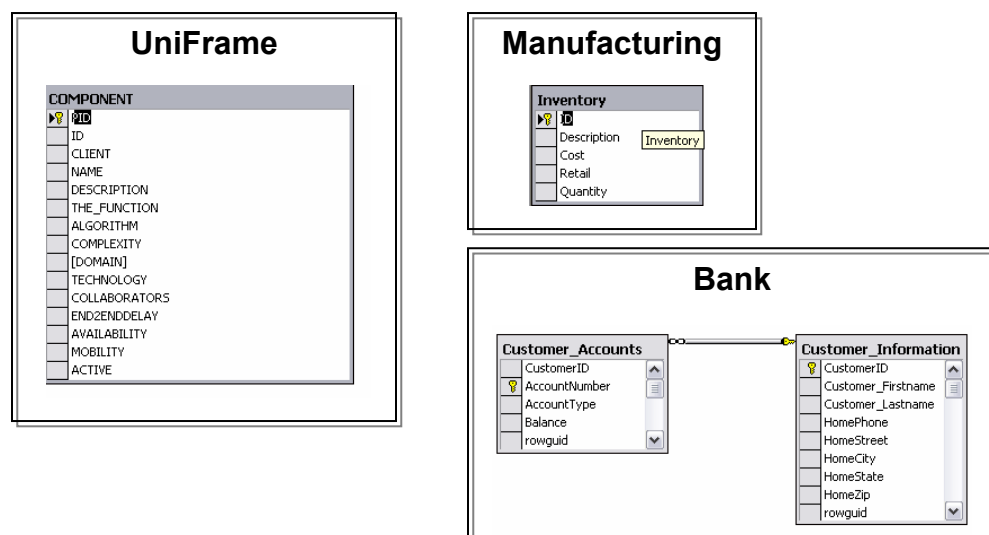


Figure 3.5 Class Diagrams for Database Objects

There are three main schemas – UniFrame, Manufacturing, and Bank, but the UniFrame schema is maintained on separate MS SQL 2000 servers than the Manufacturing and Inventory schemas. As mentioned above, all the schemas access permissions are controlled through the Security component in MS SQL 2000 so separate tables were not necessary to maintain the security permissions within the UniFrame UNWSDS.

The UniFrame schema consists of the COMPONENT table. The COMPONENT table serves as storage for the Headhunter meta-data and consists of the component information necessary for UNWSDS querying and client binding.

The Manufacturing schema consists of the INVENTORY table. The INVENTORY table serves as storage for inventory data useful to the Manufacturing domain. This schema is used for testing the prototype.

The Finance schema consists of the CUSTOMER_ACCOUNTS and CUSTOMER_INFORMATION tables. CUSTOMER_ACCOUNTS and CUSTOMER_INFORMATION tables serve as storage for bank data useful to the Financial domain. This schema is used for testing the prototype.

3.3.2.4.3 User interface

The user interface determines the presentation of the prototype to the client or user. For the presentation, the ASP.NET pages are used for dynamic generation of query responses and the HTML pages are used for static content.

The following are the ASP.NET and HTML pages which represent the user interface of the prototype in the order the user would see them: UniFrameIndex.aspx (Figure 3.6), UniFrameQuery.htm (Figure 3.7), ComponentList.aspx (Figure 3.8), GetCustomerAccountsClient.aspx (Figure 3.9), GetInventoryAccountsClient.aspx (Figure

3.10), BankDataSvc.asmx (Figure 3.11), InventoryDataSvc.asmx (Figure 3.12), ComponentControl.aspx (Figure 3.13). ComponentControl.aspx would only be used by the administrator of the prototype to be used for execution examples.



Figure 3.6 UniFrameIndex.aspx

Figure 3.7 UniFrameQuery.htm

In Figure 3.6, the UniFrameIndex.aspx page is the entry page for a user. Any accessible functions are displayed on this page. In the background, access to this page

initiates the UniFrame UNWSDS. In Figure 3.7, the UniFrameQuery.htm provides the interface to the user to select the criteria to search for active Web services.

UniFrame Search Criteria

Comments:
This web page takes user input from UniFrameQuery.htm and is processed by Headhunters to find the requested components.

Component Details: <ul style="list-style-type: none"> Domain = Finance Component Name = <u>Unselected</u> Component Description = <u>Unselected</u> Function Names = <u>Unselected</u> 	Function Attributes: <ul style="list-style-type: none"> Desired Algorithms = <u>Unselected</u> Desired Complexity = <u>Unselected</u> Technology = <u>ASP.NET</u>
Auxiliary Attributes: <ul style="list-style-type: none"> Mobility = <u>No</u> 	QOS Metrics: <ul style="list-style-type: none"> End To End Delay = <u>Unselected</u> Availability = <u>Unselected</u>

URDS Search Results

Component Search Location #1 (Laptop)

Web Service Access Point	Service Client	Name	Description
http://149.166.34.252/iberbeco/UniFrame/WebServices/BankDataSvc_3.asmx	http://149.166.34.252/iberbeco/UniFrame/WebServices/GetCustomerAccountsClient.aspx	BankDataSvc_3.asmx	Provides An Bank Account Management System

Figure 3.8 ComponentList.aspx

UniFrame
FRAMEWORK FOR SEAMLESS INTEROPERATION OF HETEROGENEOUS DISTRIBUTED SOFTWARE COMPONENTS

Get Customer Accounts Client

Comments:
Client consumes the GetCustomerAccounts detail function in the BankDataSvc web service using a web service proxy. Results are displayed and editable.

Account #	Account Number	Account Type	Balance	Account Holder Firstname	Account Holder Lastname
Edit b91a47ef-ef29-4b8a-a386-1347e136be36	1112233330	Checking	100	Joe	Doe
Edit b91a47ef-ef29-4b8a-a386-1347e136be36	1112233331	Savings	50	Joe	Doe
Edit 815b6166-a06c-4234-bf3a-2f46e5097ab1	2223344440	Checking	1000	John	Smith
Edit 815b6166-a06c-4234-bf3a-2f46e5097ab1	2223344441	Savings	50000	John	Smith
Edit 4abc7210-ba9a-4e40-b5be-942094e3774d	2311414111	Savings	23667	Steve	Shoe
Edit 2fc23c43-501e-468c-8f5e-4f0948c7d7d8	3434343430	Checking	2340	Jim	John
Edit 420d88be-31cd-44a1-aa44-bfe819a3a116	5656565660	Checking	1003	Robert	Devin

Figure 3.9 GetCustomerAccountsClient.aspx

In Figure 3.8, the ComponentList.aspx page displays the results of the Web service search criteria entered by the user. In Figure 3.9, the GetCustomerAccountsClient.aspx provides the client interface to the BankDataSvc Web

service for data manipulation and viewing. In Figure 3.10, the GetInventoryAccountsClient.aspx provides the client interface to the InventoryDataSvc Web service.

	Inventory #	Description	Cost	Retail	Quantity
Edit	289abd37-5788-47e1-a783-20bd9b81f57a	B Widget	45	120	65
Edit	17d43ec1-e0e8-45b6-aa32-47cb856275f9	A Widget	40	101	56
Edit	781bb2ae-59d0-4741-86d8-64c1ce51f5e6	C Widget	50	130	100
Edit	b390834f-b7f3-49eb-806a-81210599e514	D Widget	56	140	10
Edit	98c3eb45-795a-4099-ba1d-8a5910f3a60f	E Widget	60	150	58

Figure 3.10 GetInventoryAccountsClient.aspx

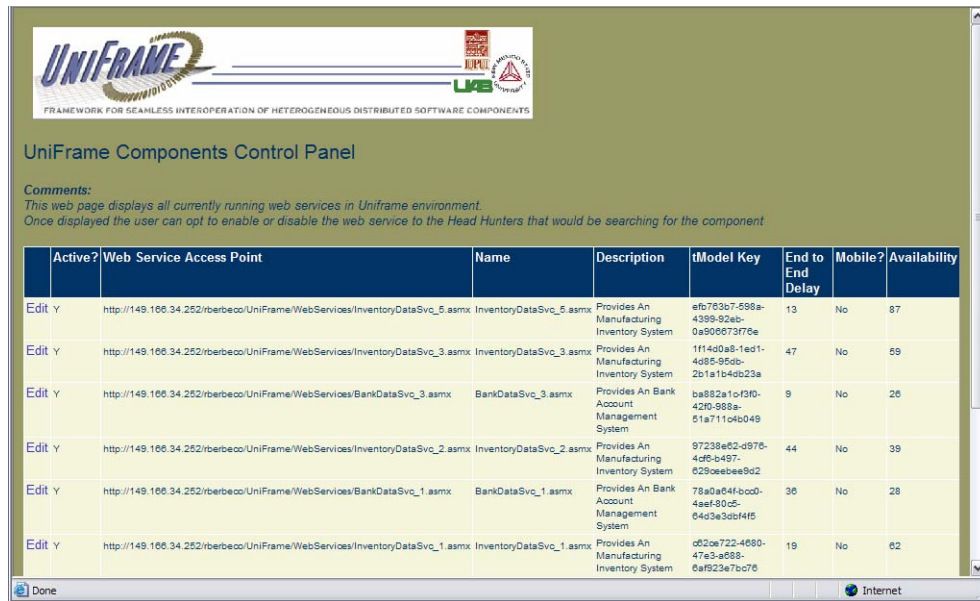
In Figure 3.11 and 3.12, the BankDataService and InventoryDataService namespaces are displayed respectively. These are the direct points of access to the Web services and the WSDL for these Web services can be accessed and viewed.

BankDataService	
The following operations are supported. For a formal definition, please review the Service Description .	
•	GetCustomerAccounts
•	UpdateCustomerAccounts

Figure 3.11 BankDataSvc.asmx

InventoryDataService	
The following operations are supported. For a formal definition, please review the Service Description .	
•	GetInventory

Figure 3.12 InventoryDataSvc.asmx



UniFrame Components Control Panel

Comments:
 This web page displays all currently running web services in UniFrame environment.
 Once displayed the user can opt to enable or disable the web service to the Head Hunters that would be searching for the component

	Active?	Web Service Access Point	Name	Description	tModel Key	End to End Delay	Mobile?	Availability
Edit	Y	http://149.166.34.252/rberbeco/UniFrame/WebServices/InventoryDataSvc_5.asmx	InventoryDataSvc_5.asmx	Provides An Manufacturing Inventory System	efb763b7-598a-4389-92eb-0a506673776e	13	No	87
Edit	Y	http://149.166.34.252/rberbeco/UniFrame/WebServices/InventoryDataSvc_3.asmx	InventoryDataSvc_3.asmx	Provides An Manufacturing Inventory System	1f14d0a8-1ed1-4e85-95db-2b1a164db23a	47	No	59
Edit	Y	http://149.166.34.252/rberbeco/UniFrame/WebServices/BankDataSvc_3.asmx	BankDataSvc_3.asmx	Provides An Bank Account Management System	ba882a10f3d0-42f0-989a-51a71104b049	9	No	26
Edit	Y	http://149.166.34.252/rberbeco/UniFrame/WebServices/InventoryDataSvc_2.asmx	InventoryDataSvc_2.asmx	Provides An Manufacturing Inventory System	97238e62-d976-4d8b-b497-6290eebee8d2	44	No	39
Edit	Y	http://149.166.34.252/rberbeco/UniFrame/WebServices/BankDataSvc_1.asmx	BankDataSvc_1.asmx	Provides An Bank Account Management System	78a0a04f-bcd0-4aa1-90d5-64d3a3cb4f5	36	No	28
Edit	Y	http://149.166.34.252/rberbeco/UniFrame/WebServices/InventoryDataSvc_1.asmx	InventoryDataSvc_1.asmx	Provides An Manufacturing Inventory System	cd20e722-4680-47e3-a688-6a923a7bc76	19	No	62

Figure 3.13 ComponentControl.aspx

In Figure 3.13, the ComponentControl.aspx page displays all the active Web services in the UNWSDS prototype. Once displayed, the administrator of the system can enable or disable the Web service to the Headhunters.

This chapter presented the implementation of the UNWSDS prototype, which serves as an extension of the URDS architecture. Since the original URDS was written in Java [4] and utilized Java RMI for its communication between components, it was not functional in .NET. Although Java can be utilized within .NET, the API is limited and RMI is not an offered capability. The UNWSDS serves as an extension of the URDS capabilities to the Microsoft .NET platform and the creation of the UNWSDS bridges the gap between Java and .NET for the UniFrame Approach.

The next chapter presents empirical experimentation that was conducted in order to validate the performance of the UNWSDS prototype.

4. VALIDATION

In order to validate the performance of the prototype, empirical experimentation was performed. Due to the limited number of Windows-OS systems available for experimentation, fewer Headhunters and clients were used than would be optimal for these experiments. Future work would include the use of more systems and Headhunters in order to evaluate the scalability of the UNWSDS.

The testing was conducted with the following: three separate clients accessing the UNWSDS via HTTP: twenty-five Headhunters – fifteen on a laptop, ten on a desktop; three IIS servers on a laptop, desktop, and embedded device in which two provided the Active Registries; fifteen total Web services available for lookup – five on a laptop, five on a desktop, and five mobile services on an embedded device; and three database servers configured as the Headhunters' meta-repositories. The experimental tests were conducted with different UniFrame search queries.

4.1 Experimentations

The below measurements were obtained being averaged over 30 requests and were conducted on PCs running Windows XP Professional and Windows 2000. The Microsoft Framework 1.1 was installed on all systems since it was used to execute some components of the UNWSDS.

The performance metric gathered during the experimentation was Average Response Time (ART), and is defined as the time taken by the UNWSDS to receive a client request, parse the request parameters, locate the appropriate services, and display the results.

Average Response Time was determined for the following three cases:

- Number of incoming queries vs. Average Response Time with all HHs activated
- Number of Web services vs. Average Response Time
- Number of HHs vs. Average Response Time

The above experiments were meant to analyze the performance of the UNWSDS under some select conditions. Due to the lack of available systems extensive scalability and fault tolerance tests were not completed. With the addition of more Windows OS systems to the UNWSDS, future experimentation would include fault tolerance and scalability.

4.2 Results

The results from the experiments are shown below as graphs. Each of the graphs is as follows in the order they appear below: Number of incoming queries vs. Average Response Time with all HHs; Number of Web services vs. Average Response Time; and Number of HHs vs. Average Response Time.

For Figure 4.1 the incoming queries were completed from one of three separate workstations. The queries themselves were sent sequentially and not all at one once, but each new query was sent to the HHs without knowing whether the query previous to it had completed yet. In this case as each query was sent an overlap could occur where one or more queries could be in the process of being serviced, while a new query is added to the system. A new query was sent to the UNWSDS prototype every one second, and the parameters used for HH searching within the query were determined randomly. The query sending time was determined arbitrarily and the parameters used for HH searching were: domain, name, description, function names, desired algorithms, desired complexity, Technology, mobility, end-to-end delay time, and availability. All experiments were conducted on a Local Area Network (LAN) connection which consisted of all the computers within the same IP subnet, same network segment, and

within the same room in the Cancer Pavilion. With the number of HHs being constant and the number of queries increasing over time the expectation is that average response time would increase. Along with the increase in the number of queries the inter-arrival time of the queries would also increase. As the number of incoming queries increases, the system becomes busy servicing those incoming requests while still servicing some requests that may not have received results yet, and as more and more incoming queries are being serviced, the average response time will continue to increase.

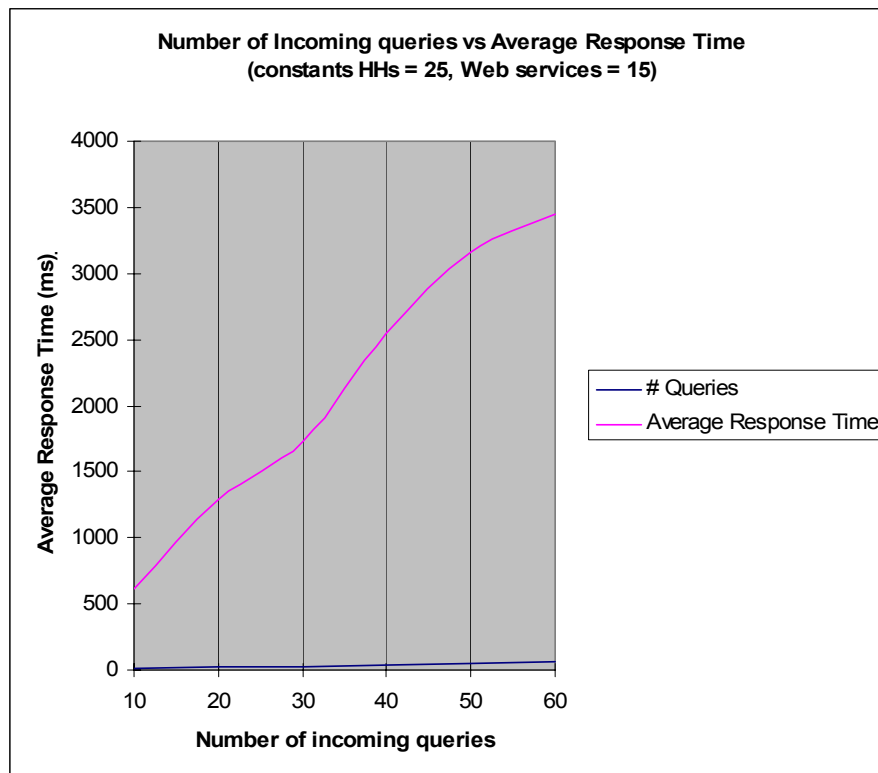


Figure 4.1 Number of incoming queries vs. Average Response Time with all HHs

Table 4.1 below represents the data obtained from the experimentation to create Figure 4.1.

# Queries	10	20	30	40	50	60
Average Response Time (ms)	614.82	1285.68	1736.11	2545.21	3156.72	3450.61

Table 4.1 Number of incoming queries vs. Average Response Time with all HHs

Figure 4.1 shows the Average Response Time with all the Headhunters active, number of Web services constant at 15, and the number of incoming UNWSDS queries increasing sequentially. By reviewing the graph it can be seen that the Average Response Time increases linearly as the initial number of queries increases, but as the number of queries continues to increase the response time continues to increase, and begins to level off somewhat. Earlier URDS experiments [4] did not specifically look at average response time for increasing number of incoming queries. Hypothetically, similar results should be obtained from the URDS, but since a similar experiment was not conducted for the URDS, an actual comparison cannot be made at this time. What is interesting to note in the graph is that the the Average Response Time dips twice from the steady linear increase. A reason for these dips occurring is that the HHs within the UNWSDS reach a point of efficiency in their ability to coordinate and handle the incoming requests. However once this threshold has been passed, the linear increase continues.

For Figure 4.2 the incoming queries were completed from one of three separate workstations sequentially as with the experiment that produced Figure 4.1. With the number of HHs and incoming queries being constant and the number of available searchable Web services within the UNWSDS increasing, the expectation is that average response time would increase. As the number of available Web services increases, the time taken for a response to be sent from HHs as to what Web services are available based on the selected criteria would increase marginally, and in turn would affect the overall response time of the system. The increase in response time from the HHs is explained by the fact that they use a database meta-repository. As the number of potential selections increases within a database query, the response time also increases. This increase in response time is more noticeable when multiple databases are queried as they are in the HHs meta-repositories in UNWSDS.

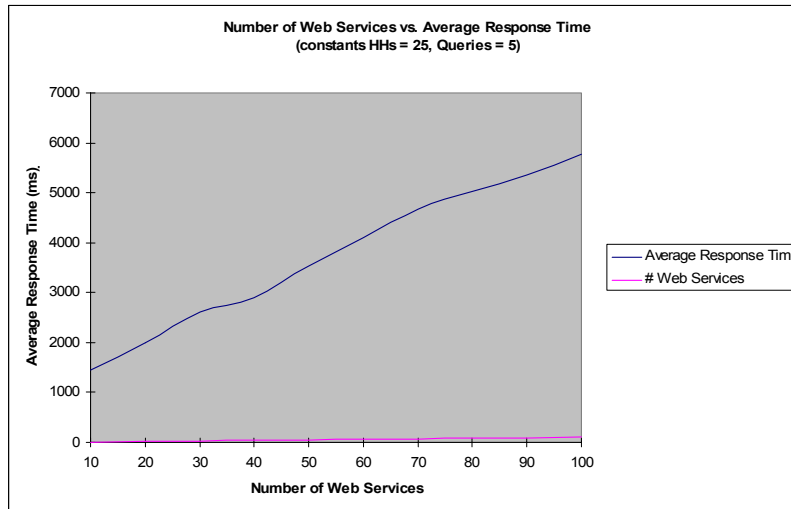


Figure 4.2 Number of Web services vs. Average Response Time

Table 4.2 below represents the data obtained from the experimentation to create Figure 4.2.

Average Response Time (ms)	1444.87	2001.03	2601.98	2902.76	3523.11	4102.32	4677.35	5023.96	5356.47	5773.54
# Web Services	10	20	30	40	50	60	70	80	90	100

Table 4.2 Number of Web services vs. Average Response Time

Figure 4.2 shows the Average Response Time with all the Headhunters active, number of incoming UNWSDS queries constant at 5, and the number of available Web services increasing sequentially. By reviewing the graph it can be seen that the Average Response Time increases linearly as the number of available Web services increases. This is in line with the expectations before the experiment was executed and experiments mentioned in [4]. As mentioned above, the increasing number of searchable and active Web services available within the UNWSDS would have an effect on the response rate of the HHs. Although the response rate would not increase dramatically it would be noticeable as large numbers of Web services were added to the UNWSDS.

For Figure 4.3 the incoming queries were completed from one of three separate workstations sequentially as with the experiment that produced Figure 4.1. With the

number of Web services being constant and the number of queries and available HHs within the UNWSDS increasing, the expectation is that average response time would increase. However, what Figure 4.3 specifically exemplifies is what effect to the average response time will occur when the number of queries continues to increase, and the number of HHs is dropped significantly. The expectation is that as the number of HHs drops the response time would decrease even as more queries are added. The decrease in HHs would offset the effect that increasing queries would have. By using Figure 4.1 as an example, with less HHs available within the UNWSDS the response time decreases (the opposite to what happened when more HHs were added in Figure 4.1).

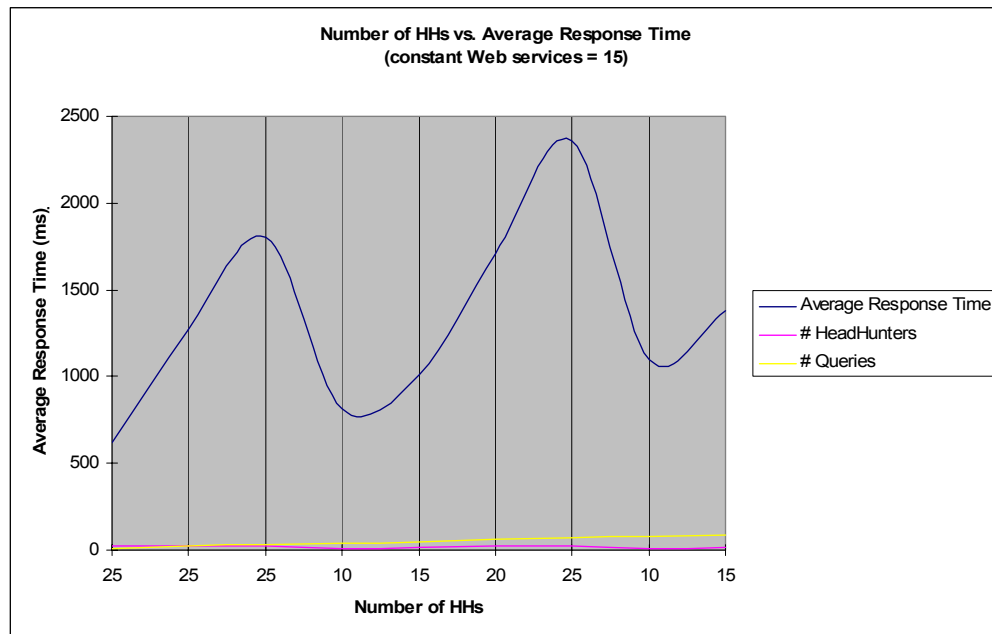


Figure 4.3 Number of HHs vs. Average Response Time

Table 4.3 below represents the data obtained from the experimentation to create Figure 4.3.

Average Response Time (ms)	617.82	1267.82	1801.21	817.12	1010.12	1707.11	2356.43	1098.44	1377.23
# HeadHunters	25	25	25	10	15	20	25	10	15
# Queries	10	20	30	40	50	60	70	80	90

Table 4.3 Number of HHs vs. Average Response Time

Figure 4.3 shows the Average Response Time with a varying number of Headhunters active, an increasing number of incoming UNWSDS queries, and a constant number of available Web services. By reviewing the graph it is seen that the Average Response Time fluctuates depending on how many incoming queries and HHs are being processed by the UNWSDS. Figure 4.2 displayed the effect that an increasing number of queries had to the UNWSDS. The response time would increase as the number of queries would increase assuming the number of HHs are held constant. However, if the number of HHs changed during the increase in incoming queries, response time would also change. What is noticed in Figure 4.3 is that if the number of HHs decreases significantly (in this case 60%), the response time would also decrease significantly. Even as more and more queries are processed, if the number of HHs is dropped, the response time also decreases. This is in line with the expectations before the experiment was executed. As mentioned above, a decrease in HHs would also decrease the response time which was observed from figure 4.1 (increasing HHs = increasing response time). Nanditha Siram's URDS experiments [4] did not specifically look at average response time for this experiment. Hypothetically, similar results should be obtained from the URDS, but since a similar experiment was not conducted for the URDS, an actual comparison cannot be made at this time.

The observations obtained from the above experimentations are summarized below:

- The number of incoming queries will increase the average response time. This observation is supported by what is observed in Figure 4.1. As the number of incoming queries increases, the system becomes busy servicing those requests while still service other requests and an increase in response time will be the result.
- The number of Web services available for Headhunter discovery will increase the average response time. This observation is supported by what is observed in Figure 4.2. This observation is also in line with URDS results in [4].

- The number of Headhunters available within the UNWSDS will increase the average response time. This observation is supported by what is observed in Figure 4.3. It is also noticed in Figure 4.3 that reducing the number of Headhunters would decrease average response time.

This chapter presented the experimental results and validation of the UNWSDS. As mentioned earlier in this chapter, the limited number of Windows-OS systems available for experimentation limited the number of experimentations that were done. With more systems added to the UNWSDS, future work would include experiments done to evaluate the scalability and fault tolerance of the UNWSDS. UNWSDS experimental results were also compared with results obtained by Nanditha Siram's URDS prototype [4]. From these comparisons, the UNWSDS prototype performance was similar to the performance of the URDS, with the average response time for both the UNWSDS and the URDS having the same results when HHs were added or subtracted.

The next chapter concludes the project by presenting the issues the UNWSDS was proposed to solve and suggests possible future extensions to this work.

5. CONCLUSION AND FUTURE WORK

.NET is a new platform for building interoperable distributed applications that can communicate over the Internet and are cross-platform interoperable.

The UNWSDS proposed in this project is based upon the URDS prototype [4], and is meant to bridge the gap from the URDS to .NET Web services. The URDS prototype was developed in Java and although Java could be used within .NET; Java RMI, which was used for communication between the different components of the URDS, could not. The UNWSDS prototype can be located on desktops, laptops, and embedded devices either on a wired or wireless network, and utilize .NET Web services for dynamic discovery, service consumption, and for a user friendly web interface. Once the UNWSDS prototype was completed, testing was done to evaluate its efficiency within the .NET and compact frameworks and compared with the original URDS prototype.

The contributions of this project are:

- The creation of an architecture, the UNWSDS, which is based upon the URDS prototype [4]. The UNWSDS extends the capabilities of the UniFrame Approach to the Microsoft .NET platform.
- Implementation of a UNWSDS prototype based on the .NET model.
- Validation of the UNWSDS by experimentations and detailed case study.

Future work to complete for the UNWSDS involves enhancing the implementation of the prototype and adding more UNWSDS Web services to the prototype for further testing.

Some future work for the UNWSDS and the prototype includes:

- One major restriction in the current prototype is that it appears CE IIS is in a Beta form. It is expected that, as CE IIS becomes a more seasoned and supported Web server package, the embedded device functionality can be tested more with .NET. Web services cannot currently run as applications on an embedded device.
- Embedded devices still use embedded programming languages and are not currently leveraging .NET. With the full release of CE.NET this is expected to change.
- Connecting the Headhunter meta-repositories through heterogeneous merge replication publisher/subscriber communication.
- Adding more systems to the UNWSDS for further scalability and fault tolerance testing.
- Enhanced support for link failure detection.

In conclusion, this project has presented the UniFrame .NET Web Service Discovery Service (UNWSDS), which facilitates the expansion of the UniFrame Approach and extends the URDS to the Microsoft .NET platform. As the use of Web services becomes more prolific over the Internet, a strong need exists for an effective and efficient method of discovery for clients to use these services. The UNWSDS, coupled with the UniFrame Approach, represents a promising method for the discovery of Web services that are geographically scattered.

LIST OF REFERENCES

- [1] cs.iupui.edu/uniFrame, “UniFrame Meta-Component Model for Distributed Systems”, <http://www.cs.iupui.edu/uniFrame/description.html>
- [2] Jeffrey Richter, *Applied Microsoft .NET Framework Programming*, Microsoft Press, Washington, 2002.
- [3] Raje, R., Auguston, M., Bryant, B. R., Olson, A., Burt, C., “A Unified Approach for the Integration of Distributed Heterogeneous Software Components”, Proceedings of the Monterey Workshop on Engineering Automation for Software Intensive System Integration, 2001, pp. 109-119.
- [4] Nanditha N. Siram, "An Architecture for Discovery of Heterogeneous Software Components", MS Thesis, Department of Computer and Information Science, Indiana University Purdue University Indianapolis, March 2002.
- [5] Universal Description, Discovery and Integration of Web Services, “UDDI Technical White Paper”, September 2000, http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf
- [6] Yasser Shohoud, *Real World XML Web Services*, Addison Wesley, Boston, 2003
- [7] IBM, “Web services – the Web’s next revolution”, 2001, <http://www-105.ibm.com/developerworks/education.nsf/webservices-onlinecourse-bytitle/BA84142372686CFB862569A400601C18?OpenDocument>
- [8] DevX, “Building Web Services with .NET”, 2001, <http://archive.devx.com/dotnet/articles/cp0901/cp0901-1/waws.asp>

- [9] Universal Description, Discovery and Integration of Web Services, “UDDI V3 Specification”, July 2002, http://uddi.org/pubs/uddi_v3.htm
- [10] ONJava, “An Introduction to WSIL”, October 2002, <http://www.onjava.com/pub/a/onjava/2002/10/16/wsil.html>
- [11] Microsoft Corporation, “Web Services Discovery Tool (Disco.exe)”, 2002, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpgrfwebservicediscoverytooldiscoexe.asp>
- [12] Internet Journal, “Web Servers: What is an application server?”, 2002, <http://www.intranetjournal.com/faqs/webserver/appserver.html>
- [13] Microsoft Corporation, “What is the .NET Framework?”, May 2003, <http://www.microsoft.com/net/basics/framework.asp>
- [14] The Data Administration Newsletter, “Doing –dot- NET right”, 2002, <http://www.tdan.com/i020hy04.htm>
- [15] Web Developer’s Virtual Library, “What is .NET?”, 2000, http://www.wdvl.com/Authoring/ASP/NetRev/what_is.html
- [16] IISFAQ, “What is IIS?”, 2003, <http://www.iisfaq.com/default.aspx?View=A409>
- [17] Microsoft Corporation, “IIS Lockdown Tool 2.1”, 2002, <http://www.microsoft.com/downloads/details.aspx?FamilyID=dde9efc0-bb30-47eb-9a61-fd755d23cdec&DisplayLang=en>

APPENDIX

Source Code

Styles.css	
/* Filename: Styles.css Created Date: 3/28/03 Author: Bob Berbeco Description: CSS Stylesheet for the Uniframe Project */	
BODY	{ font-weight: normal; font-size: 10pt; word-spacing: normal; text-transform: none; color: #003366; font-family: Arial; letter-spacing: normal; background-color: #999965; }
H1, H2, H3, H4, H5, TH, THEAD, TFOOT	{ COLOR: #003366; }
H1	{ font-family: Arial, Helvetica, sans-serif; font-size: 2em; font-weight: 700; font-style: normal; text-decoration: none; word-spacing: normal; letter-spacing: normal; text-transform: none; }
H2	{ font-family: Arial, Helvetica, sans-serif; font-size: 1.75em; font-weight: 700; font-style: normal; text-decoration: none; word-spacing: normal; letter-spacing: normal; text-transform: none; }
H3	{ font-family: Arial, Helvetica, sans-serif;

	<pre> font-size: 1.58em; font-weight: 500; font-style: normal; text-decoration: none; word-spacing: normal; letter-spacing: normal; text-transform: none; } </pre>
H4	<pre> { font-family: Arial, Helvetica, sans-serif; font-size: 1.33em; font-weight: 500; text-decoration: none; word-spacing: normal; letter-spacing: normal; text-transform: none; } </pre>
H5, DT	<pre> { font-family: Arial, Helvetica, sans-serif; font-size: 1em; font-weight: 700; font-style: normal; text-decoration: none; word-spacing: normal; letter-spacing: normal; text-transform: none; } </pre>
H6	<pre> { font-family: Arial, Helvetica, sans-serif; font-size: .8em; font-weight: 700; font-style: normal; text-decoration: none; word-spacing: normal; letter-spacing: normal; text-transform: none; } </pre>
TFOOT, THEAD	<pre> { font-size: 1em; word-spacing: normal; letter-spacing: normal; text-transform: none; font-family: Arial, Helvetica, sans-serif; } </pre>
TH	<pre> { vertical-align: baseline; font-size: 1em; font-weight: bold; word-spacing: normal; letter-spacing: normal; text-transform: none; } </pre>

```

        font-family: Arial, Helvetica, sans-serif;
    }

A:link {
    text-decoration: none;
    color: #3333cc;
}

A:visited {
    text-decoration: none;
    color: #333399;
}

A:active {
    text-decoration: none;
    color: #333399;
}

A:hover {
    text-decoration: underline;
    color: #3333cc;
}

SMALL {
    font-size: .7em;
}

BIG {
    font-size: 1.17em;
}

BLOCKQUOTE, PRE {
    font-family: Courier New, monospace;
}

UL LI {
    list-style-type: square ;
}

UL LI LI {
    list-style-type: disc;
}

UL LI LI LI {
    list-style-type: circle;
}

OL LI {
    list-style-type: decimal;
}

OL OL LI {
    list-style-type: lower-alpha;
}

```

```
OL OL OL LI {
    list-style-type: lower-roman;
}
```

```
IMG {
    margin-top: 5px;
    margin-left: 10px;
    margin-right: 10px;
}
```

```
TABLE
{
    color: maroon;
    font-family: Arial;
    font-weight: normal;
    font-size: 10pt;
}
```

```
TD
{
    color: #003366;
    font-family: Arial;
    font-weight: normal;
    font-size: 10pt;
    vertical-align: baseline;
}
```

```
TR
{
    color: maroon;
    font-family: Arial;
    font-weight: normal;
    font-size: 10pt;
}
```

Web.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <system.web>
    <identity impersonate="true" userName="uniframe" password="uniframe" />

    <!-- DYNAMIC DEBUG COMPILATION
      Set compilation debug="true" to insert debugging symbols (.pdb information)
      into the compiled page. Because this creates a larger file that executes
      more slowly, you should set this value to true only when debugging and to
      false at all other times. For more information, refer to the documentation about
      debugging ASP.NET files.
    -->
    <compilation defaultLanguage="vb" debug="true" />

    <!-- CUSTOM ERROR MESSAGES
      Set customErrors mode="On" or "RemoteOnly" to enable custom error messages, "Off" to disable.
      Add <error> tags for each of the errors you want to handle.
```

```

-->
<customErrors mode="RemoteOnly" />

<!-- AUTHENTICATION
    This section sets the authentication policies of the application. Possible modes are "Windows",
    "Forms", "Passport" and "None"
-->
<authentication mode="Windows" />

<!-- AUTHORIZATION
    This section sets the authorization policies of the application. You can allow or deny access
    to application resources by user or role. Wildcards: "*" mean everyone, "?" means anonymous
    (unauthenticated) users.
-->
<authorization>
    <allow users="*" /> <!-- Allow all users -->

    <!-- <allow    users="[comma separated list of users]"
        roles="[comma separated list of roles]"/>
        <deny     users="[comma separated list of users]"
        roles="[comma separated list of roles]"/>
    -->
</authorization>

<!-- APPLICATION-LEVEL TRACE LOGGING
    Application-level tracing enables trace log output for every page within an application.
    Set trace enabled="true" to enable application trace logging. If pageOutput="true", the
    trace information will be displayed at the bottom of each page. Otherwise, you can view the
    application trace log by browsing the "trace.axd" page from your web application
    root.
-->
<trace enabled="false" requestLimit="10" pageOutput="false" traceMode="SortByTime"
localOnly="true" />

<!-- SESSION STATE SETTINGS
    By default ASP.NET uses cookies to identify which requests belong to a particular session.
    If cookies are not available, a session can be tracked by adding a session identifier to the URL.
    To disable cookies, set sessionState cookieless="true".
-->
<sessionState
    mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;user id=sa;password="
    cookieless="false"
    timeout="20"
/>

<!-- GLOBALIZATION
    This section sets the globalization settings of the application.
-->
<globalization requestEncoding="utf-8" responseEncoding="utf-8" />

</system.web>

```


[illegible]

Components.vb

```
Option Strict Off
Option Explicit On

Imports System
Imports System.Data
Imports System.Runtime.Serialization
Imports System.Xml

<Serializable(), _
System.ComponentModel.DesignerCategoryAttribute("code"), _
System.Diagnostics.DebuggerStepThrough(), _
System.ComponentModel.ToolboxItem(true)> _
Public Class Components
    Inherits DataSet

    Private tableCOMPONENT As COMPONENTDataTable

    Public Sub New()
        MyBase.New
        Me.InitClass
        Dim schemaChangedHandler As System.ComponentModel.CollectionChangeEventHandler =
AddressOf Me.SchemaChanged
        AddHandler Me.Tables.CollectionChanged, schemaChangedHandler
        AddHandler Me.Relations.CollectionChanged, schemaChangedHandler
    End Sub

    Protected Sub New(ByVal info As SerializationInfo, ByVal context As StreamingContext)
        MyBase.New
        Dim strSchema As String = CType(info.GetValue("XmlSchema", GetType(System.String)),String)
        If (Not (strSchema) Is Nothing) Then
            Dim ds As DataSet = New DataSet
            ds.ReadXmlSchema(New XmlTextReader(New System.IO.StringReader(strSchema)))
            If (Not (ds.Tables("COMPONENT")) Is Nothing) Then
                Me.Tables.Add(New COMPONENTDataTable(ds.Tables("COMPONENT")))
            End If
            Me.DataSetName = ds.DataSetName
            Me.Prefix = ds.Prefix
            Me.Namespace = ds.Namespace
            Me.Locale = ds.Locale
        End Sub
    End Sub
End Class
```

```

        Me.CaseSensitive = ds.CaseSensitive
        Me.EnforceConstraints = ds.EnforceConstraints
        Me.Merge(ds, false, System.Data.MissingSchemaAction.Add)
        Me.InitVars
    Else
        Me.InitClass
    End If
    Me.GetSerializationData(info, context)
    Dim schemaChangedHandler As System.ComponentModel.CollectionChangeEventHandler =
AddressOf Me.SchemaChanged
    AddHandler Me.Tables.CollectionChanged, schemaChangedHandler
    AddHandler Me.Relations.CollectionChanged, schemaChangedHandler
End Sub

<System.ComponentModel.Browsable(false), _
System.ComponentModel.DesignerSerializationVisibilityAttribute(System.ComponentModel.DesignerSer-
ializationVisibility.Content)> _
    Public ReadOnly Property COMPONENT As COMPONENTDataTable
        Get
            Return Me.tableCOMPONENT
        End Get
    End Property

    Public Overrides Function Clone() As DataSet
        Dim cln As Components = CType(MyBase.Clone, Components)
        cln.InitVars
        Return cln
    End Function

    Protected Overrides Function ShouldSerializeTables() As Boolean
        Return false
    End Function

    Protected Overrides Function ShouldSerializeRelations() As Boolean
        Return false
    End Function

    Protected Overrides Sub ReadXmlSerializable(ByVal reader As XmlReader)
        Me.Reset
        Dim ds As DataSet = New DataSet
        ds.ReadXml(reader)
        If (Not (ds.Tables("COMPONENT")) Is Nothing) Then
            Me.Tables.Add(New COMPONENTDataTable(ds.Tables("COMPONENT")))
        End If
        Me.DataSetName = ds.DataSetName
        Me.Prefix = ds.Prefix
        Me.Namespace = ds.Namespace
        Me.Locale = ds.Locale
        Me.CaseSensitive = ds.CaseSensitive
        Me.EnforceConstraints = ds.EnforceConstraints
        Me.Merge(ds, false, System.Data.MissingSchemaAction.Add)
        Me.InitVars
    End Sub

    Protected Overrides Function GetSchemaSerializable() As System.Xml.Schema.XmlSchema

```

```

    Dim stream As System.IO.MemoryStream = New System.IO.MemoryStream
    Me.WriteXmlSchema(New XmlTextWriter(stream, Nothing))
    stream.Position = 0
    Return System.Xml.Schema.XmlSchema.Read(New XmlTextReader(stream), Nothing)
End Function

Friend Sub InitVars()
    Me.tableCOMPONENT = CType(Me.Tables("COMPONENT"),COMPONENTDataTable)
    If (Not (Me.tableCOMPONENT) Is Nothing) Then
        Me.tableCOMPONENT.InitVars
    End If
End Sub

Private Sub InitClass()
    Me.DataSetName = "Components"
    Me.Prefix = ""
    Me.Namespace = "http://www.tempuri.org/Components.xsd"
    Me.Locale = New System.Globalization.CultureInfo("en-US")
    Me.CaseSensitive = false
    Me.EnforceConstraints = true
    Me.tableCOMPONENT = New COMPONENTDataTable
    Me.Tables.Add(Me.tableCOMPONENT)
End Sub

Private Function ShouldSerializeCOMPONENT() As Boolean
    Return false
End Function

Private Sub SchemaChanged(ByVal sender As Object, ByVal e As
System.ComponentModel.CollectionChangeEventArgs)
    If (e.Action = System.ComponentModel.CollectionChangeAction.Remove) Then
        Me.InitVars
    End If
End Sub

Public Delegate Sub COMPONENTRowChangeEventHandler(ByVal sender As Object, ByVal e As
COMPONENTRowChangeEvent)

<System.Diagnostics.DebuggerStepThrough()> _
Public Class COMPONENTDataTable
    Inherits DataTable
    Implements System.Collections.IEnumerable

    Private columnPID As DataColumn

    Private columnID As DataColumn

    Private columnNAME As DataColumn

    Private columnDESCRIPTION As DataColumn

    Private columnTHE_FUNCTION As DataColumn

    Private columnALGORITHM As DataColumn

    Private columnCOMPLEXITY As DataColumn

```

```

Private columnDOMAIN As DataColumn

Private columnTECHNOLOGY As DataColumn

Private columnCOLLABORATORS As DataColumn

Private columnEND2ENDDELAY As DataColumn

Private columnAVAILABILITY As DataColumn

Private columnMOBILITY As DataColumn

Private columnACTIVE As DataColumn

Friend Sub New()
    MyBase.New("COMPONENT")
    Me.InitClass
End Sub

Friend Sub New(ByVal table As DataTable)
    MyBase.New(table.TableName)
    If (table.CaseSensitive <> table.DataSet.CaseSensitive) Then
        Me.CaseSensitive = table.CaseSensitive
    End If
    If (table.Locale.ToString <> table.DataSet.Locale.ToString) Then
        Me.Locale = table.Locale
    End If
    If (table.Namespace <> table.DataSet.Namespace) Then
        Me.Namespace = table.Namespace
    End If
    Me.Prefix = table.Prefix
    Me.MinimumCapacity = table.MinimumCapacity
    Me.DisplayExpression = table.DisplayExpression
End Sub

<System.ComponentModel.Browsable(false)> _
Public ReadOnly Property Count As Integer
    Get
        Return Me.Rows.Count
    End Get
End Property

Friend ReadOnly Property PIDColumn As DataColumn
    Get
        Return Me.columnPID
    End Get
End Property

Friend ReadOnly Property IDColumn As DataColumn
    Get
        Return Me.columnID
    End Get
End Property

Friend ReadOnly Property NAMEColumn As DataColumn

```

```
Get
    Return Me.columnNAME
End Get
End Property

Friend ReadOnly Property DESCRIPTIONColumn As DataColumn
Get
    Return Me.columnDESCRIPTION
End Get
End Property

Friend ReadOnly Property THE_FUNCTIONColumn As DataColumn
Get
    Return Me.columnTHE_FUNCTION
End Get
End Property

Friend ReadOnly Property ALGORITHMColumn As DataColumn
Get
    Return Me.columnALGORITHM
End Get
End Property

Friend ReadOnly Property COMPLEXITYColumn As DataColumn
Get
    Return Me.columnCOMPLEXITY
End Get
End Property

Friend ReadOnly Property DOMAINColumn As DataColumn
Get
    Return Me.columnDOMAIN
End Get
End Property

Friend ReadOnly Property TECHNOLOGYColumn As DataColumn
Get
    Return Me.columnTECHNOLOGY
End Get
End Property

Friend ReadOnly Property COLLABORATORSColumn As DataColumn
Get
    Return Me.columnCOLLABORATORS
End Get
End Property

Friend ReadOnly Property END2ENDDELAYColumn As DataColumn
Get
    Return Me.columnEND2ENDDELAY
End Get
End Property

Friend ReadOnly Property AVAILABILITYColumn As DataColumn
Get
    Return Me.columnAVAILABILITY
```

```

    End Get
End Property

Friend ReadOnly Property MOBILITYColumn As DataColumn
    Get
        Return Me.columnMOBILITY
    End Get
End Property

Friend ReadOnly Property ACTIVEColumn As DataColumn
    Get
        Return Me.columnACTIVE
    End Get
End Property

Public Default ReadOnly Property Item(ByVal index As Integer) As COMPONENTRow
    Get
        Return CType(Me.Rows(index),COMPONENTRow)
    End Get
End Property

Public Event COMPONENTRowChanged As COMPONENTRowChangeEventHandler

Public Event COMPONENTRowChanging As COMPONENTRowChangeEventHandler

Public Event COMPONENTRowDeleted As COMPONENTRowChangeEventHandler

Public Event COMPONENTRowDeleting As COMPONENTRowChangeEventHandler

Public Overloads Sub AddCOMPONENTRow(ByVal row As COMPONENTRow)
    Me.Rows.Add(row)
End Sub

Public Overloads Function AddCOMPONENTRow(ByVal PID As System.Guid, ByVal ID As String,
ByVal NAME As String, ByVal DESCRIPTION As String, ByVal THE_FUNCTION As String, ByVal
ALGORITHM As String, ByVal COMPLEXITY As String, ByVal DOMAIN As String, ByVal
TECHNOLOGY As String, ByVal COLLABORATORS As String, ByVal END2ENDDELAY As String,
ByVal AVAILABILITY As String, ByVal MOBILITY As String, ByVal ACTIVE As String) As
COMPONENTRow
    Dim rowCOMPONENTRow As COMPONENTRow = CType(Me.NewRow,COMPONENTRow)
    rowCOMPONENTRow.ItemArray = New Object() {PID, ID, NAME, DESCRIPTION,
THE_FUNCTION, ALGORITHM, COMPLEXITY, DOMAIN, TECHNOLOGY, COLLABORATORS,
END2ENDDELAY, AVAILABILITY, MOBILITY, ACTIVE}
    Me.Rows.Add(rowCOMPONENTRow)
    Return rowCOMPONENTRow
End Function

Public Function FindByPID(ByVal PID As System.Guid) As COMPONENTRow
    Return CType(Me.Rows.Find(New Object() {PID}),COMPONENTRow)
End Function

Public Function GetEnumerator() As System.Collections.IEnumerator Implements
System.Collections.IEnumerable.GetEnumerator
    Return Me.Rows.GetEnumerator
End Function

```

```

Public Overrides Function Clone() As DataTable
    Dim cln As COMPONENTDataTable = CType(MyBase.Clone, COMPONENTDataTable)
    cln.InitVars
    Return cln
End Function

```

```

Protected Overrides Function CreateInstance() As DataTable
    Return New COMPONENTDataTable
End Function

```

```

Friend Sub InitVars()
    Me.columnPID = Me.Columns("PID")
    Me.columnID = Me.Columns("ID")
    Me.columnNAME = Me.Columns("NAME")
    Me.columnDESCRIPTION = Me.Columns("DESCRIPTION")
    Me.columnTHE_FUNCTION = Me.Columns("THE_FUNCTION")
    Me.columnALGORITHM = Me.Columns("ALGORITHM")
    Me.columnCOMPLEXITY = Me.Columns("COMPLEXITY")
    Me.columnDOMAIN = Me.Columns("DOMAIN")
    Me.columnTECHNOLOGY = Me.Columns("TECHNOLOGY")
    Me.columnCOLLABORATORS = Me.Columns("COLLABORATORS")
    Me.columnEND2ENDDELAY = Me.Columns("END2ENDDELAY")
    Me.columnAVAILABILITY = Me.Columns("AVAILABILITY")
    Me.columnMOBILITY = Me.Columns("MOBILITY")
    Me.columnACTIVE = Me.Columns("ACTIVE")
End Sub

```

```

Private Sub InitClass()
    Me.columnPID = New DataColumn("PID", GetType(System.Guid), Nothing,
System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnPID)
    Me.columnID = New DataColumn("ID", GetType(System.String), Nothing,
System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnID)
    Me.columnNAME = New DataColumn("NAME", GetType(System.String), Nothing,
System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnNAME)
    Me.columnDESCRIPTION = New DataColumn("DESCRIPTION", GetType(System.String),
Nothing, System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnDESCRIPTION)
    Me.columnTHE_FUNCTION = New DataColumn("THE_FUNCTION", GetType(System.String),
Nothing, System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnTHE_FUNCTION)
    Me.columnALGORITHM = New DataColumn("ALGORITHM", GetType(System.String),
Nothing, System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnALGORITHM)
    Me.columnCOMPLEXITY = New DataColumn("COMPLEXITY", GetType(System.String),
Nothing, System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnCOMPLEXITY)
    Me.columnDOMAIN = New DataColumn("DOMAIN", GetType(System.String), Nothing,
System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnDOMAIN)
    Me.columnTECHNOLOGY = New DataColumn("TECHNOLOGY", GetType(System.String),
Nothing, System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnTECHNOLOGY)
    Me.columnCOLLABORATORS = New DataColumn("COLLABORATORS",

```



```

GetType(System.String), Nothing, System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnCOLLABORATORS)
    Me.columnEND2ENDDELAY = New DataColumn("END2ENDDELAY",
GetType(System.String), Nothing, System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnEND2ENDDELAY)
    Me.columnAVAILABILITY = New DataColumn("AVAILABILITY", GetType(System.String),
Nothing, System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnAVAILABILITY)
    Me.columnMOBILITY = New DataColumn("MOBILITY", GetType(System.String), Nothing,
System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnMOBILITY)
    Me.columnACTIVE = New DataColumn("ACTIVE", GetType(System.String), Nothing,
System.Data.MappingType.Element)
    Me.Columns.Add(Me.columnACTIVE)
    Me.Constraints.Add(New UniqueConstraint("Constraint1", New DataColumn() {Me.columnPID},
true))
    Me.columnPID.AllowDBNull = false
    Me.columnPID.Unique = true
    Me.columnID.AllowDBNull = false
End Sub

Public Function NewCOMPONENTRow() As COMPONENTRow
    Return CType(Me.NewRow,COMPONENTRow)
End Function

Protected Overrides Function NewRowFromBuilder(ByVal builder As DataRowBuilder) As DataRow
    Return New COMPONENTRow(builder)
End Function

Protected Overrides Function GetRowType() As System.Type
    Return GetType(COMPONENTRow)
End Function

Protected Overrides Sub OnRowChanged(ByVal e As DataRowChangeEventArgs)
    MyBase.OnRowChanged(e)
    If (Not (Me.COMPONENTRowChangedEvent) Is Nothing) Then
        RaiseEvent COMPONENTRowChanged(Me, New
COMPONENTRowChangeEvent(CType(e.Row,COMPONENTRow), e.Action))
    End If
End Sub

Protected Overrides Sub OnRowChanging(ByVal e As DataRowChangeEventArgs)
    MyBase.OnRowChanging(e)
    If (Not (Me.COMPONENTRowChangingEvent) Is Nothing) Then
        RaiseEvent COMPONENTRowChanging(Me, New
COMPONENTRowChangeEvent(CType(e.Row,COMPONENTRow), e.Action))
    End If
End Sub

Protected Overrides Sub OnRowDeleted(ByVal e As DataRowChangeEventArgs)
    MyBase.OnRowDeleted(e)
    If (Not (Me.COMPONENTRowDeletedEvent) Is Nothing) Then
        RaiseEvent COMPONENTRowDeleted(Me, New
COMPONENTRowChangeEvent(CType(e.Row,COMPONENTRow), e.Action))
    End If
End Sub

```

```

Protected Overrides Sub OnRowDeleting(ByVal e As DataRowChangeEventArgs)
    MyBase.OnRowDeleting(e)
    If (Not (Me.COMPONENTRowDeletingEvent) Is Nothing) Then
        RaiseEvent COMPONENTRowDeleting(Me, New
COMPONENTRowChangeEvent(CType(e.Row,COMPONENTRow), e.Action))
    End If
End Sub

Public Sub RemoveCOMPONENTRow(ByVal row As COMPONENTRow)
    Me.Rows.Remove(row)
End Sub
End Class

<System.Diagnostics.DebuggerStepThrough(> _
Public Class COMPONENTRow
    Inherits DataRow

    Private tableCOMPONENT As COMPONENTDataTable

    Friend Sub New(ByVal rb As DataRowBuilder)
        MyBase.New(rb)
        Me.tableCOMPONENT = CType(Me.Table,COMPONENTDataTable)
    End Sub

    Public Property PID As System.Guid
        Get
            Return CType(Me(Me.tableCOMPONENT.PIDColumn),System.Guid)
        End Get
        Set
            Me(Me.tableCOMPONENT.PIDColumn) = value
        End Set
    End Property

    Public Property ID As String
        Get
            Return CType(Me(Me.tableCOMPONENT.IDColumn),String)
        End Get
        Set
            Me(Me.tableCOMPONENT.IDColumn) = value
        End Set
    End Property

    Public Property NAME As String
        Get
            Try
                Return CType(Me(Me.tableCOMPONENT.NAMEColumn),String)
            Catch e As InvalidCastException
                Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
            End Try
        End Get
        Set
            Me(Me.tableCOMPONENT.NAMEColumn) = value
        End Set
    End Property

```

```

Public Property DESCRIPTION As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.DESRIPTIONColumn),String)
        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableCOMPONENT.DESRIPTIONColumn) = value
    End Set
End Property

Public Property THE_FUNCTION As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.THE_FUNCTIONColumn),String)
        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableCOMPONENT.THE_FUNCTIONColumn) = value
    End Set
End Property

Public Property ALGORITHM As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.ALGORITHMColumn),String)
        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableCOMPONENT.ALGORITHMColumn) = value
    End Set
End Property

Public Property COMPLEXITY As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.COMPLEXITYColumn),String)
        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableCOMPONENT.COMPLEXITYColumn) = value
    End Set
End Property

Public Property DOMAIN As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.DOMAINColumn),String)

```

```

        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableCOMPONENT.DOMAINColumn) = value
    End Set
End Property

Public Property TECHNOLOGY As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.TECHNOLOGYColumn),String)
        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableCOMPONENT.TECHNOLOGYColumn) = value
    End Set
End Property

Public Property COLLABORATORS As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.COLLABORATORSColumn),String)
        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableCOMPONENT.COLLABORATORSColumn) = value
    End Set
End Property

Public Property END2ENDDELAY As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.END2ENDDELAYColumn),String)
        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableCOMPONENT.END2ENDDELAYColumn) = value
    End Set
End Property

Public Property AVAILABILITY As String
    Get
        Try
            Return CType(Me(Me.tableCOMPONENT.AVAILABILITYColumn),String)
        Catch e As InvalidCastException
            Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
        End Try
    End Get

```

```

Set
    Me(Me.tableCOMPONENT.AVAILABILITYColumn) = value
End Set
End Property

Public Property MOBILITY As String
Get
    Try
        Return CType(Me(Me.tableCOMPONENT.MOBILITYColumn),String)
    Catch e As InvalidCastException
        Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
    End Try
End Get
Set
    Me(Me.tableCOMPONENT.MOBILITYColumn) = value
End Set
End Property

Public Property ACTIVE As String
Get
    Try
        Return CType(Me(Me.tableCOMPONENT.ACTIVEColumn),String)
    Catch e As InvalidCastException
        Throw New StrongTypingException("Cannot get value because it is DBNull.", e)
    End Try
End Get
Set
    Me(Me.tableCOMPONENT.ACTIVEColumn) = value
End Set
End Property

Public Function IsNAMENull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.NAMEColumn)
End Function

Public Sub SetNAMENull()
    Me(Me.tableCOMPONENT.NAMEColumn) = System.Convert.DBNull
End Sub

Public Function IsDESCRIPTIONNull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.DESRIPTIONColumn)
End Function

Public Sub SetDESCRIPTIONNull()
    Me(Me.tableCOMPONENT.DESRIPTIONColumn) = System.Convert.DBNull
End Sub

Public Function IsTHE_FUNCTIONNull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.THE_FUNCTIONColumn)
End Function

Public Sub SetTHE_FUNCTIONNull()
    Me(Me.tableCOMPONENT.THE_FUNCTIONColumn) = System.Convert.DBNull
End Sub

Public Function IsALGORITHMNull() As Boolean

```

```

Return Me.IsNull(Me.tableCOMPONENT.ALGORITHMColumn)
End Function

Public Sub SetALGORITHMNull()
    Me(Me.tableCOMPONENT.ALGORITHMColumn) = System.Convert.DBNull
End Sub

Public Function IsCOMPLEXITYNull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.COMPLEXITYColumn)
End Function

Public Sub SetCOMPLEXITYNull()
    Me(Me.tableCOMPONENT.COMPLEXITYColumn) = System.Convert.DBNull
End Sub

Public Function IsDOMAINNull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.DOMAINColumn)
End Function

Public Sub SetDOMAINNull()
    Me(Me.tableCOMPONENT.DOMAINColumn) = System.Convert.DBNull
End Sub

Public Function IsTECHNOLOGYNull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.TECHNOLOGYColumn)
End Function

Public Sub SetTECHNOLOGYNull()
    Me(Me.tableCOMPONENT.TECHNOLOGYColumn) = System.Convert.DBNull
End Sub

Public Function IsCOLLABORATORSNull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.COLLABORATORSColumn)
End Function

Public Sub SetCOLLABORATORSNull()
    Me(Me.tableCOMPONENT.COLLABORATORSColumn) = System.Convert.DBNull
End Sub

Public Function IsEND2ENDDELAYNull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.END2ENDDELAYColumn)
End Function

Public Sub SetEND2ENDDELAYNull()
    Me(Me.tableCOMPONENT.END2ENDDELAYColumn) = System.Convert.DBNull
End Sub

Public Function IsAVAILABILITYNull() As Boolean
    Return Me.IsNull(Me.tableCOMPONENT.AVAILABILITYColumn)
End Function

Public Sub SetAVAILABILITYNull()
    Me(Me.tableCOMPONENT.AVAILABILITYColumn) = System.Convert.DBNull
End Sub

Public Function IsMOBILITYNull() As Boolean

```

```

        Return Me.IsNull(Me.tableCOMPONENT.MOBILITYColumn)
    End Function

    Public Sub SetMOBILITYNull()
        Me(Me.tableCOMPONENT.MOBILITYColumn) = System.Convert.DBNull
    End Sub

    Public Function IsACTIVENull() As Boolean
        Return Me.IsNull(Me.tableCOMPONENT.ACTIVEColumn)
    End Function

    Public Sub SetACTIVENull()
        Me(Me.tableCOMPONENT.ACTIVEColumn) = System.Convert.DBNull
    End Sub
End Class

<System.Diagnostics.DebuggerStepThrough()> _
Public Class COMPONENTRowChangeEvent
    Inherits EventArgs

    Private eventRow As COMPONENTRow

    Private eventAction As DataRowAction

    Public Sub New(ByVal row As COMPONENTRow, ByVal action As DataRowAction)
        MyBase.New
        Me.eventRow = row
        Me.eventAction = action
    End Sub

    Public ReadOnly Property Row As COMPONENTRow
        Get
            Return Me.eventRow
        End Get
    End Property

    Public ReadOnly Property Action As DataRowAction
        Get
            Return Me.eventAction
        End Get
    End Property
End Class
End Class

```

UniFrameQuery.htm

```

<%@ Page Language="vb" AutoEventWireup="false" Codebehind="UniFrameQuery.aspx.vb"
Inherits="UniFrame.UniFrameQuery" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!--
Filename:          UniFrameQuery.htm
Created Date:      3/28/03
Author:           Bob Berbeco
Description:       This web page takes user input from form and passes it to
                   ComponentList.aspx for component searching via Head Hunters

```

Modifications: 4/13/03 Bob Berbeco - added comments

```
-->
<html>
  <head>
    <title>UniFrameQuery</title>
    <meta content="Microsoft Visual Studio.NET 7.0" name="GENERATOR">
    <meta content="Visual Basic 7.0" name="CODE_LANGUAGE">
    <meta content="JavaScript" name="vs_defaultClientScript">
    <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
    <meta http-equiv="PRAGMA" content="NO-CACHE">
    <LINK href="Styles.css" type="text/css" rel="stylesheet">
  </head>
  <body MS_POSITIONING="GridLayout">
    <form id="queryFilterForm" action="ComponentList.aspx" method="post"
runat="server">
      <p align="left"><IMG height="96" src="gfx\uniframeheader.gif" width="520">
      </p>
      <H3>UniFrame Query Form</H3>
      <p><i><b>Comments:</b></i><br>
        This web page takes user input from form and passes it to
ComponentList.aspx
        for component searching via Head Hunters.<br>
        Enter search criteria and click Search UniFrame.</i></p>
      <table cellSpacing="1" cellPadding="1" width="774" border="0">
        <tr>
          <!-- Search by component details section -->
          <td bgColor="#cccccc" colSpan="5" height="19">
            <b>Search By Component Details</b>
          </td>
          <td width="5"></td>
        </tr>
        <tr>
          <td width="104" height="0"></td>
          <td width="137"></td>
          <td width="60"></td>
          <td width="277"></td>
          <td width="184"></td>
          <td width="5"></td>
        </tr>
        <tr>
          <td colSpan="2" height="21">Domain</td>
          <td colSpan="4" height="21"><select name="domain">
            <option value="Finance"
selected>Finance</option>
            <option
value="Manufacturing">Manufacturing</option>
          </select>
          </td>
        </tr>
        <tr>
          <td colSpan="2" height="37">Component Name
<small><i>(Enter Keywords)</i></small></td>
          <td colSpan="3" height="37">
            <input type="text" size="80"
name="componentName">

```



```

RMI</option>
value="CORBA">CORBA</option>
value="Voyager">Voyager</option>
</select>
</td>
</tr>
<tr>
<td bgColor="#cccccc" colSpan="5" height="19">
<!-- Search by auxillary attributes section -->
<b>Search By Auxillary Attributes</b>
</td>
<td width="5"></td>
</tr>
<tr>
<td colSpan="2" height="30">Mobility</td>
<td colSpan="4"><select name="mobility">
<option value="No" selected>No</option>
<option value="Yes">Yes</option>
</select>
</td>
</tr>
<tr>
<td colspan="5">
<!-- Search by QOS Metrics section -->
<td bgColor="#cccccc" colSpan="5" height="19">
<b>Search By QOS Metrics</b></td>
<td width="5"></td>
</tr>
<tr>
<td width="104" bgColor="#cccccc" height="15">Select</td>
<td bgColor="#cccccc" colSpan="2">QOS Parameter</td>
<td width="600" bgColor="#cccccc">Constraints
<small><i>(* can be used for a range -
such as 4* = 40-49)</small></i></td>
<td width="5"></td>
</tr>
<tr>
<td width="104" height="41">
<input type="checkbox" value="end2endDelay"
name="qosMetric">
</td>
<td colSpan="2">End-to-end Delay
</td>
<td width="277">
<input type="text" size="50"
name="end2endDelayValue">
</td>
<td width="184">
</td>
<td width="5"></td>
</tr>
<tr>
<td width="104" height="41">
<input type="checkbox" value="availability"

```

```

name="qosMetric">
                                </td>
                                <td colSpan="2">Availability</td>
                                <td width="277">
                                    <input type="text" size="50"
name="availabilityValue">
                                </td>
                                <td width="184">
                                </td>
                                <td width="5"></td>
                                </tr>
                                <tr>
                                </tr>
                                </tr>
                                </table>
                                <p align="center">
                                    <input type="submit" name="SubmitForm" value="Search
UniFrame"> <input type="reset" name="Submit2" value="Reset Form">
                                </p>
                                </form>
                                </body>
</html>

```

Headhunters.asmx.vb

```

'Filename:          Headhunters.asmx.vb
'Created Date:      3/28/03
'Author:            Bob Berbeco
'Description:       This file is the main web service for the UniFrame Headhunters.
'                   The web service refreshes the HeadHunter meta-data tables
'                   based on the following algorithm:
'                   1) User inputs query
'                   2) Query is started
'                   3) Each HeadHunter looks in their directory for active web service
'                      components
'                   4) Each HeadHunter finds components and adds them to their meta-data
'                      tables (database)
'                   5) Once Headhunters have completed their meta-data refresh, the
'                      actual query starts
'                   There are three UniFrame Headhunters being utilized
'                   Components are housed on a desktop, laptop, and embedded device
'Modifications:     4/13/03 Bob Berbeco - added comments
Imports System.Web.Services
Imports System.Data
Imports System.Data.SqlClient
Imports System.IO
Imports System

<WebService(Namespace:="http://localhost/rberbeco/UniFrame/Headhunters")> _
Public Class Headhunters
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

```

'This call is required by the Web Services Designer.
InitializeComponent()

'Add your own initialization code after the InitializeComponent() call

End Sub

'Required by the Web Services Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Web Services Designer

'It can be modified using the Web Services Designer.

'Do not modify it using the code editor.

<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
 components = New System.ComponentModel.Container()

End Sub

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)

'CODEGEN: This procedure is required by the Web Services Designer

'Do not modify it using the code editor.

If disposing Then

 If Not (components Is Nothing) Then

 components.Dispose()

 End If

End If

MyBase.Dispose(disposing)

End Sub

#End Region

<WebMethod()> Public Function FindWebServices()

 ' ** Declare all variables **

 ' HeadHunter1 variables

 Dim myConnectionHH1 As SqlConnection

 Dim myCommandHH1 As SqlCommand

 Dim strWebServicesDirectoryHH1 As String

 Dim strWebServicesURLHH1 As String

 Dim strFilesHH1 As String()

 Dim strFileHH1 As String

 Dim dirStoreFileHH1 As Directory

 ' HeadHunter2 variables

 Dim myConnectionHH2 As SqlConnection

 Dim myCommandHH2 As SqlCommand

 Dim strWebServicesDirectoryHH2 As String

 Dim strWebServicesURLHH2 As String

 Dim strFilesHH2 As String()

 Dim strFileHH2 As String

 Dim dirStoreFileHH2 As Directory

 ' HeadHunter3 variables

 Dim myConnectionHH3 As SqlConnection

 Dim myCommandHH3 As SqlCommand

```

Dim strWebServicesDirectoryHH3 As String
Dim strWebServicesURLHH3 As String
Dim strFilesHH3 As String()
Dim strFileHH3 As String
Dim dirStoreFileHH3 As Directory

' Declare the remaining variables
Dim strFilename As String
Dim strComponent As String
Dim strDescription As String
Dim strThe_Function As String
Dim strAlgorithm As String
Dim strComplexity As String
Dim strDomain As String
Dim strCollaborators As String
Dim strEnd2EndDelay As String
Dim strAvailability As String
Dim strMobility As String
Dim strClient As String

' ** HeadHunter search and refresh data tables subroutines **
' NOTE: For the sake of the experimentation it is important
' keep Headhunters query web services separate from
' other Headhunters

' ** HeadHunter1 search and update meta-data subroutine **

' Initialize the search directory and bind URL
strWebServicesDirectoryHH1 = _
"C:\Inetpub\wwwroot\rberbeco\UniFrame\WebServices\"
strWebServicesURLHH1 = _
"http://149.166.34.252/rberbeco/UniFrame/WebServices/"

' Initialize SQL connect
myConnectionHH1 = _
New SqlConnection("User
Id=HeadHunter_1;Password=HeadHunter1;database=UniFrame;server=IN-CANC-867126\MP")
myConnectionHH1.Open()
myCommandHH1 = New SqlCommand("DELETE FROM Component", myConnectionHH1)
myCommandHH1.ExecuteNonQuery()
myConnectionHH1.Close()

' Find and process web services
strFilesHH1 = dirStoreFileHH1.GetFiles(strWebServicesDirectoryHH1, "*.asmx")
For Each strFileHH1 In strFilesHH1
    strFilename = Mid(strFileHH1, (strWebServicesDirectoryHH1.Length + 1))
    If (InStr(strFilename, "Mobile") > 0) Then
        strMobility = "Yes"
        strWebServicesURLHH1 = _
"http://162.1.223.223/rberbeco/UniFrame/WebServices/"
        strFilename = String.Concat(Mid(strFilename, 2, (strFilename.Length - 5)), "XML")
    Else
        strMobility = "No"
    End If

```

```

strComponent = String.Concat(strWebServicesURLHH1, strFilename)
strAvailability = CInt(Int((100 * Rnd()) + 1))
strEnd2EndDelay = CInt(Int((50 * Rnd()) + 1))
If (InStr(strFilename, "Bank") > 0) Then
    strDescription = "Provides An Bank Account Management System"
    strThe_Function = "Acts As An Account Server"
    strAlgorithm = "Complete Simple Bank Transfers"
    strComplexity = "O(1)"
    strDomain = "Finance"
    strCollaborators = "AccountClient"
    If (strMobility = "No") Then
        strClient = String.Concat(strWebServicesURLHH1, "GetCustomerAccountsClient.aspx")
    Else
        strClient = String.Concat("http://162.1.223.223/rberbeco/UniFrame/WebServices/",
strFilename)
    End If
Else
    strDescription = "Provides An Manufacturing Inventory System"
    strThe_Function = "Acts As An Inventory Server"
    strAlgorithm = "Complete Inventory Transfers"
    strComplexity = "O(n)"
    strDomain = "Manufacturing"
    strCollaborators = "ManufacturingClient"
    If (strMobility = "No") Then
        strClient = String.Concat(strWebServicesURLHH1, "GetInventoryAccountsClient.aspx")
    Else
        strClient = String.Concat("http://162.1.223.223/rberbeco/UniFrame/WebServices/",
strFilename)
    End If
End If
myConnectionHH1.Open()
myCommandHH1 = New SqlCommand("INSERT INTO Component " & _
    "(ID,NAME,DESCRIPTION,THE_FUNCTION,ALGORITHM,COMPLEXITY" & _
    ",DOMAIN,COLLABORATORS,CLIENT, END2ENDDELAY, AVAILABILITY, _
MOBILITY) VALUES " & _
    "(" & strComponent & _
    "," & strFilename & _
    "," & strDescription & _
    "," & strThe_Function & _
    "," & strAlgorithm & _
    "," & strComplexity & _
    "," & strDomain & _
    "," & strCollaborators & _
    "," & strClient & _
    "," & strEnd2EndDelay & _
    "," & strAvailability & _
    "," & strMobility & _
    ")", myConnectionHH1)
myCommandHH1.ExecuteNonQuery()
myConnectionHH1.Close()
Next

' ** HeadHunter2 search and update meta-data subroutine **

' Initialize the search directory and bind URL
strWebServicesDirectoryHH2 = _

```

```

"\\IN-CANC-829868\WebServices\$"
strWebServicesURLHH2 = _
"http://149.166.34.253/rberbeco/UniFrame/WebServices/"

' Initialize SQL connect
myConnectionHH2 = _
New SqlConnection("User
Id=HeadHunter_2;Password=HeadHunter2;database=UniFrame;server=IN-CANC-829868\MP")
myConnectionHH2.Open()
myCommandHH2 = New SqlCommand("DELETE FROM Component", myConnectionHH2)
myCommandHH2.ExecuteNonQuery()
myConnectionHH2.Close()

' Find and process web services
strFilesHH2 = dirStoreFileHH2.GetFiles(strWebServicesDirectoryHH2, "*.asmx")
For Each strFileHH2 In strFilesHH2
    strFilename = Mid(strFileHH2, (strWebServicesDirectoryHH2.Length + 1))
    If (InStr(strFilename, "Mobile") > 0) Then
        strMobility = "Yes"
        strWebServicesURLHH2 = _
        "http://162.1.223.223/rberbeco/UniFrame/WebServices/"
        strFilename = String.Concat(Mid(strFilename, 2, (strFilename.Length - 5)), "XML")
    Else
        strMobility = "No"
    End If
    strComponent = String.Concat(strWebServicesURLHH2, strFilename)
    strAvailability = CInt(Int((100 * Rnd()) + 1))
    strEnd2EndDelay = CInt(Int((50 * Rnd()) + 1))
    If (InStr(strFilename, "Bank") > 0) Then
        strDescription = "Provides An Bank Account Management System"
        strThe_Function = "Acts As An Account Server"
        strAlgorithm = "Complete Simple Bank Transfers"
        strComplexity = "O(1)"
        strDomain = "Finance"
        strCollaborators = "AccountClient"
        If (strMobility = "No") Then
            strClient = String.Concat(strWebServicesURLHH2, "GetCustomerAccountsClient.aspx")
        Else
            strClient = String.Concat("http://162.1.223.223/rberbeco/UniFrame/WebServices/",
strFilename)
        End If
    Else
        strDescription = "Provides An Manufacturing Inventory System"
        strThe_Function = "Acts As An Inventory Server"
        strAlgorithm = "Complete Inventory Transfers"
        strComplexity = "O(n)"
        strDomain = "Manufacturing"
        strCollaborators = "ManufacturingClient"
        If (strMobility = "No") Then
            strClient = String.Concat(strWebServicesURLHH2, "GetInventoryAccountsClient.aspx")
        Else
            strClient = String.Concat("http://162.1.223.223/rberbeco/UniFrame/WebServices/",
strFilename)
        End If
    End If
    myConnectionHH2.Open()

```

```

myCommandHH2 = New SqlCommand("INSERT INTO Component " & _
    "(ID,NAME,DESCRIPTION,THE_FUNCTION,ALGORITHM,COMPLEXITY" & _
    ",DOMAIN,COLLABORATORS,CLIENT, END2ENDDelay, AVAILABILITY, _
MOBILITY) VALUES " & _
    "(" & strComponent & _
    "," & strFilename & _
    "," & strDescription & _
    "," & strThe_Function & _
    "," & strAlgorithm & _
    "," & strComplexity & _
    "," & strDomain & _
    "," & strCollaborators & _
    "," & strClient & _
    "," & strEnd2EndDelay & _
    "," & strAvailability & _
    "," & strMobility & _
    ")", myConnectionHH2)
myCommandHH2.ExecuteNonQuery()
myConnectionHH2.Close()
Next

' ** HeadHunter3 search and update meta-data subroutine **

' Initialize the search directory and bind URL
strWebServicesDirectoryHH3 = _
"C:\Inetpub\wwwroot\rberbeco\UniFrame\WebServices\Mobile"
strWebServicesURLHH3 = _
"http://149.166.34.252/rberbeco/UniFrame/WebServices/Mobile"

' Initialize SQL connect
myConnectionHH3 = _
New SqlConnection("User
Id=HeadHunter_3;Password=HeadHunter3;database=UniFrame;server=IN-CANC-829868\MP2")
myConnectionHH3.Open()
myCommandHH3 = New SqlCommand("DELETE FROM Component", myConnectionHH3)
myCommandHH3.ExecuteNonQuery()
myConnectionHH3.Close()

' Find and process web services
strFilesHH3 = dirStoreFileHH3.GetFiles(strWebServicesDirectoryHH3, "*.asmx")
For Each strFileHH3 In strFilesHH3
    strFilename = Mid(strFileHH3, (strWebServicesDirectoryHH3.Length + 1))
    If (InStr(strFilename, "Mobile") > 0) Then
        strMobility = "Yes"
        strWebServicesURLHH3 = _
            "http://162.1.223.223/rberbeco/UniFrame/WebServices/"
        strFilename = String.Concat(Mid(strFilename, 2, (strFilename.Length - 5)), "XML")
    Else
        strMobility = "No"
    End If
    strComponent = String.Concat(strWebServicesURLHH3, strFilename)
    strAvailability = CInt(Int((100 * Rnd()) + 1))
    strEnd2EndDelay = CInt(Int((50 * Rnd()) + 1))
    If (InStr(strFilename, "Bank") > 0) Then
        strDescription = "Provides An Bank Account Management System"
    End If
End For

```



```

        strThe_Function = "Acts As An Account Server"
        strAlgorithm = "Complete Simple Bank Transfers"
        strComplexity = "O(1)"
        strDomain = "Finance"
        strCollaborators = "AccountClient"
        If (strMobility = "No") Then
            strClient = String.Concat(strWebServicesURLHH1, "GetCustomerAccountsClient.aspx")
        Else
            strClient = String.Concat("http://162.1.223.223/rberbeco/UniFrame/WebServices/",
strFilename)
        End If
    Else
        strDescription = "Provides An Manufacturing Inventory System"
        strThe_Function = "Acts As An Inventory Server"
        strAlgorithm = "Complete Inventory Transfers"
        strComplexity = "O(n)"
        strDomain = "Manufacturing"
        strCollaborators = "ManufacturingClient"
        If (strMobility = "No") Then
            strClient = String.Concat(strWebServicesURLHH1, "GetInventoryAccountsClient.aspx")
        Else
            strClient = String.Concat("http://162.1.223.223/rberbeco/UniFrame/WebServices/",
strFilename)
        End If
    End If
    myConnectionHH3.Open()
    myCommandHH3 = New SqlCommand("INSERT INTO Component " & _
        "(ID,NAME,DESCRIPTION,THE_FUNCTION,ALGORITHM,COMPLEXITY" & _
        ",DOMAIN,COLLABORATORS,CLIENT, END2ENDDELAY, AVAILABILITY,
MOBILITY) VALUES " & _
        "(" & strComponent & _
        "," & strFilename & _
        "," & strDescription & _
        "," & strThe_Function & _
        "," & strAlgorithm & _
        "," & strComplexity & _
        "," & strDomain & _
        "," & strCollaborators & _
        "," & strClient & _
        "," & strEnd2EndDelay & _
        "," & strAvailability & _
        "," & strMobility & _
        ")", myConnectionHH3)
    myCommandHH3.ExecuteNonQuery()
    myConnectionHH3.Close()
Next
End Function
End Class

```

ComponentList.aspx

```

<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System" %>

```

```

<%@ Import Namespace="UniFrame.Headhunters" %>
<%@ Page Language="vb" %>
<!--
Filename:          ComponentList.aspx
Created Date:      3/28/03
Author:           Bob Berbeco
Description:       This web page takes user input from UniFrameQuery form
                   and processes it for component searching via Head Hunters
                   Results are displayed via data grid
Modifications:    4/13/03 Bob Berbeco - added comments
-->
<HTML>
    <HEAD>
        <title>URDS Search Results</title>
        <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
        <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
        <LINK href="Styles.css" type="text/css" rel="stylesheet">
        <script runat="server">

            'Page load subroutine
            Sub Page_Load(sender As Object, e As EventArgs)
            If Not Page.IsPostBack Then
                BindGrid()
            End If
            End Sub

            'BindGrid subroutine for data grid configuration
            Private Sub BindGrid()

                'Declare variables used for head hunter query
                Dim domain as String
                Dim componentName as String
                Dim componentDescription as String
                Dim functionNames as String
                Dim algorithms as String
                Dim complexity as String
                Dim technology as String
                Dim availabilityValue as string
                Dim end2endDelayValue as string
                Dim mobility as string

                'Create HH instances of connection and command object for database
connections
                Dim myConnectionHH1 As SqlConnection = _
                    New SqlConnection("User
Id=HeadHunter_1;Password=HeadHunter1;database=UniFrame;server=IN-CANC-867126\MP")
                Dim myCommandHH1 As SqlCommand = _
                    New SqlCommand("sp_search_for_components", myConnectionHH1)
                Dim myConnectionHH2 As SqlConnection = _
                    New SqlConnection("User
Id=HeadHunter_2;Password=HeadHunter2;database=UniFrame;server=IN-CANC-829868\MP")
                Dim myCommandHH2 As SqlCommand = _
                    New SqlCommand("sp_search_for_components", myConnectionHH2)

```

```

Dim myConnectionHH3 As SqlConnection = _
    New SqlConnection("User
Id=HeadHunter_3;Password=HeadHunter3;database=UniFrame;server=IN-CANC-829868\MP2")
Dim myCommandHH3 As SqlCommand = _
    New SqlCommand("sp_search_for_components", myConnectionHH3)

'Mark the HH commands as a stored procedures since the
'UniFrame database stored procedure will be invoked
myCommandHH1.CommandType = CommandType.StoredProcedure
myCommandHH2.CommandType = CommandType.StoredProcedure
myCommandHH3.CommandType = CommandType.StoredProcedure

'**Add all parameters to be processed by HH stored procedures**

'Domain
if (Request.Form("domain") = "")
    domainLabel.Text = "Unselected"
else
    domainLabel.Text = Request.Form("domain")
end if
domain = "%" & Request.Form("domain") & "%"
myCommandHH1.Parameters.Add(New SqlClient.SqlParameter("@domain", _
System.Data.SqlDbType.VarChar, 30)).Value = domain
myCommandHH2.Parameters.Add(New SqlClient.SqlParameter("@domain", _
System.Data.SqlDbType.VarChar, 30)).Value = domain
myCommandHH3.Parameters.Add(New SqlClient.SqlParameter("@domain", _
System.Data.SqlDbType.VarChar, 30)).Value = domain

'componentName
if (Request.Form("componentName") = "")
    componentNameLabel.Text = "Unselected"
else
    componentNameLabel.Text = Request.Form("componentName")
end if
componentName = "%" & Request.Form("componentName") & "%"
myCommandHH1.Parameters.Add(New
SqlClient.SqlParameter("@componentName", _
System.Data.SqlDbType.VarChar, 100)).Value = componentName
myCommandHH2.Parameters.Add(New
SqlClient.SqlParameter("@componentName", _
System.Data.SqlDbType.VarChar, 100)).Value = componentName
myCommandHH3.Parameters.Add(New
SqlClient.SqlParameter("@componentName", _
System.Data.SqlDbType.VarChar, 100)).Value = componentName

'componentDescription
if (Request.Form("componentDescription") = "")
    componentDescriptionLabel.Text = "Unselected"
else
    componentDescriptionLabel.Text =
Request.Form("componentDescription")
end if
componentDescription = "%" & Request.Form("componentDescription") & "%"
myCommandHH1.Parameters.Add(New
SqlClient.SqlParameter("@componentDescription", _

```

```

        System.Data.SqlDbType.VarChar, 1000)).Value = componentDescription
myCommandHH2.Parameters.Add(New
SqlCommand.SqlParameter("@componentDescription", _
        System.Data.SqlDbType.VarChar, 1000)).Value = componentDescription
myCommandHH3.Parameters.Add(New
SqlCommand.SqlParameter("@componentDescription", _
        System.Data.SqlDbType.VarChar, 1000)).Value = componentDescription

'functionNames
if (Request.Form("functionNames") = "")
    functionNamesLabel.Text = "Unselected"
else
    functionNamesLabel.Text = Request.Form("functionNames")
end if
functionNames = "%" & Request.Form("functionNames") & "%"
myCommandHH1.Parameters.Add(New
SqlCommand.SqlParameter("@functionNames", _
        System.Data.SqlDbType.VarChar, 500)).Value = functionNames
myCommandHH2.Parameters.Add(New
SqlCommand.SqlParameter("@functionNames", _
        System.Data.SqlDbType.VarChar, 500)).Value = functionNames
myCommandHH3.Parameters.Add(New
SqlCommand.SqlParameter("@functionNames", _
        System.Data.SqlDbType.VarChar, 500)).Value = functionNames

'algorithms
if (Request.Form("algorithms") = "")
    algorithmsLabel.Text = "Unselected"
else
    algorithmsLabel.Text = Request.Form("algorithms")
end if
algorithms = "%" & Request.Form("algorithms") & "%"
myCommandHH1.Parameters.Add(New
SqlCommand.SqlParameter("@algorithms", _
        System.Data.SqlDbType.VarChar, 200)).Value = algorithms
myCommandHH2.Parameters.Add(New
SqlCommand.SqlParameter("@algorithms", _
        System.Data.SqlDbType.VarChar, 200)).Value = algorithms
myCommandHH3.Parameters.Add(New
SqlCommand.SqlParameter("@algorithms", _
        System.Data.SqlDbType.VarChar, 200)).Value = algorithms

'complexity
if (Request.Form("complexity") = "")
    complexityLabel.Text = "Unselected"
else
    complexityLabel.Text = Request.Form("complexity")
end if
complexity = "%" & Request.Form("complexity") & "%"
myCommandHH1.Parameters.Add(New
SqlCommand.SqlParameter("@complexity", _
        System.Data.SqlDbType.VarChar, 30)).Value = complexity
myCommandHH2.Parameters.Add(New
SqlCommand.SqlParameter("@complexity", _
        System.Data.SqlDbType.VarChar, 30)).Value = complexity
myCommandHH3.Parameters.Add(New

```

```

SqlClient.SqlParameter("@complexity", _
    System.Data.SqlDbType.VarChar, 30)).Value = complexity

    'technology
    if (Request.Form("technology") = "")
        technologyLabel.Text = "Unselected"
    else
        technologyLabel.Text = Request.Form("technology")
    end if
    technology = "%" & Request.Form("technology") & "%"
    myCommandHH1.Parameters.Add(New
SqlClient.SqlParameter("@technology", _
    System.Data.SqlDbType.VarChar, 30)).Value = technology
    myCommandHH2.Parameters.Add(New
SqlClient.SqlParameter("@technology", _
    System.Data.SqlDbType.VarChar, 30)).Value = technology
    myCommandHH3.Parameters.Add(New
SqlClient.SqlParameter("@technology", _
    System.Data.SqlDbType.VarChar, 30)).Value = technology

    'availabilityValue
    if (Request.Form("availabilityValue") = "")
        availabilityLabel.Text = "Unselected"
    else
        availabilityLabel.Text = Request.Form("availabilityValue")
    end if
    availabilityValue = "%" & Request.Form("availabilityValue") & "%"
    myCommandHH1.Parameters.Add(New
SqlClient.SqlParameter("@availabilityValue", _
    System.Data.SqlDbType.VarChar, 30)).Value = availabilityValue
    myCommandHH2.Parameters.Add(New
SqlClient.SqlParameter("@availabilityValue", _
    System.Data.SqlDbType.VarChar, 30)).Value = availabilityValue
    myCommandHH3.Parameters.Add(New
SqlClient.SqlParameter("@availabilityValue", _
    System.Data.SqlDbType.VarChar, 30)).Value = availabilityValue

    'end2endDelayValue
    if (Request.Form("end2endDelayValue") = "")
        end2endDelayValueLabel.Text = "Unselected"
    else
        end2endDelayValueLabel.Text =
Request.Form("end2endDelayValue")
    end if
    end2endDelayValue = "%" & Request.Form("end2endDelayValue") & "%"
    myCommandHH1.Parameters.Add(New
SqlClient.SqlParameter("@end2endDelayValue", _
    System.Data.SqlDbType.VarChar, 30)).Value = end2endDelayValue
    myCommandHH2.Parameters.Add(New
SqlClient.SqlParameter("@end2endDelayValue", _
    System.Data.SqlDbType.VarChar, 30)).Value = end2endDelayValue
    myCommandHH3.Parameters.Add(New
SqlClient.SqlParameter("@end2endDelayValue", _
    System.Data.SqlDbType.VarChar, 30)).Value = end2endDelayValue

    'mobility

```

```

        if (Request.Form("mobility") = "")
            mobilityLabel.Text = "Unselected"
        else
            mobilityLabel.Text = Request.Form("mobility")
        end if
        mobility = "%" & Request.Form("mobility") & "%"
        myCommandHH1.Parameters.Add(New SqlClient.SqlParameter("@mobility", _
        System.Data.SqlDbType.VarChar, 5)).Value = mobility
        myCommandHH2.Parameters.Add(New SqlClient.SqlParameter("@mobility", _
        System.Data.SqlDbType.VarChar, 5)).Value = mobility
        myCommandHH3.Parameters.Add(New SqlClient.SqlParameter("@mobility", _
        System.Data.SqlDbType.VarChar, 5)).Value = mobility

        'Execute all HH stored procedures with above data,
        'add results to results data grid objects
        Try

            'Open all HH meta-data connections, execute stored procedures
            myConnectionHH1.Open()
            DG_Components1.DataSource =
myCommandHH1.ExecuteReader(CommandBehavior.CloseConnection)
            DG_Components1.DataBind()
            myConnectionHH2.Open()
            DG_Components2.DataSource =
myCommandHH2.ExecuteReader(CommandBehavior.CloseConnection)
            DG_Components2.DataBind()
            myConnectionHH3.Open()
            DG_Components3.DataSource =
myCommandHH3.ExecuteReader(CommandBehavior.CloseConnection)

            DG_Components3.DataBind()

            'Catch any result grids that have no results to notify user
            if (DG_Components1.Items.Count = 0) then
                DataGridIsNullLabel1.Text = "Head Hunters have completed
their individual search routines and could not discover web services with requested parameters on location
#1."
            end if
            if (DG_Components2.Items.Count = 0) then
                DataGridIsNullLabel2.Text = "Head Hunters have completed
their individual search routines and could not discover web services with requested parameters on location
#2."
            end if
            if (DG_Components3.Items.Count = 0) then
                DataGridIsNullLabel3.Text = "Head Hunters have completed
their individual search routines and could not discover web services with requested parameters on location
#3."
            end if

            'Catch general SQL error
            Catch SQLexc As SqlException
                Response.Write("Error occured while Generating Data. Error is " &
SQLexc.ToString())
            End Try
        End Sub
</script>

```

```

</HEAD>
<body>
  <!-- Output HTML all HH data grid results -->
  <h3 align="center">UniFrame Search Criteria</h3>
  <p><i><b>Comments:</b><br>
    This web page takes user input from UniFrameQuery.htm and
    is processed by Headhunters
    to find the requested components.</i></p>

  <!-- Show the user selected parameters -->
  <table cellSpacing="1" cellPadding="5" border="1" align="center" width="75%"
  bgcolor="#ffffe5">
    <tr>
      <td><b>Component Details:</b>
        <ul>
          <li>
            Domain =
            <asp:label id="domainLabel" runat="server"
            Font-Underline="true"></asp:label>
          <li>
            Component Name =
            <asp:label id="componentNameLabel"
            runat="server" Font-Underline="true"></asp:label>
          <li>
            Component Description =
            <asp:label id="componentDescriptionLabel"
            runat="server" Font-Underline="true"></asp:label>
          <li>
            Function Names =
            <asp:label id="functionNamesLabel"
            runat="server" Font-Underline="true"></asp:label></li></ul>
        </td>
      <td><b>Function Attributes:</b>
        <ul>
          <li>
            Desired Algorithms =
            <asp:label id="algorithmsLabel"
            runat="server" Font-Underline="true"></asp:label>
          <li>
            Desired Complexity =
            <asp:label id="complexityLabel"
            runat="server" Font-Underline="true"></asp:label>
          <li>
            Technology =
            <asp:label id="technologyLabel"
            runat="server" Font-Underline="true"></asp:label></li></ul>
        </td>
    </tr>
    <tr>
      <td width="365"><b>Auxillary Attributes:</b>
        <ul>
          <li>
            Mobility =
            <asp:label id="mobilityLabel"
            runat="server" Font-Underline="true"></asp:label></li></ul>
        </td>
    </tr>
  </table>

```

```

        <td><b>QOS Metrics:</b>
        <ul>
            <li>
                End-to-end Delay =
                <asp:Label id="end2endDelayValueLabel"
runat="server" Font-Underline="true"></asp:Label>
            </li>
            Availability =
            <asp:Label id="availabilityLabel"
runat="server" Font-Underline="true"></asp:Label></li></ul>
        </td>
    </tr>
</table>

<!-- Display results -->
<h3 align="center">URDS Search Results</h3>

<!-- Display all laptop components found by HHs -->
<h4 align="center">Component Search Location #1 (Laptop)</h4>
<form id="Form1" method="post" runat="server">
    <asp:DataGrid ID="DG_Components1" AutoGenerateColumns="False"
Width="99%" BorderColor="#000000" Runat="server">
        <HeaderStyle Font-Size="13px" Font-Names="Verdana" Font-
Bold="True" BackColor="#003366"></HeaderStyle>
        <ItemStyle Font-Size="13px" Font-Names="verdana"
BackColor="Beige"></ItemStyle>
        <Columns>
            <asp:TemplateColumn HeaderText="Web Service Access
Point" HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                <ItemTemplate>
                    <asp:Label id=Label1 Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "BINDING") %>'>
                    </asp:Label>
                </ItemTemplate>
            </asp:TemplateColumn>
            <asp:TemplateColumn HeaderText="Service Client"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                <ItemTemplate>
                    <asp:Label Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "PROXY") %>' ID="Label5" />
                </ItemTemplate>
            </asp:TemplateColumn>
            <asp:TemplateColumn HeaderText="Name" HeaderStyle-
Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                <ItemTemplate>
                    <asp:Label Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "NAME") %>' ID="Label2" />
                </ItemTemplate>
            </asp:TemplateColumn>
            <asp:TemplateColumn HeaderText="Description"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                <ItemTemplate>
                    <asp:Label Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "DESCRIPTION") %>' ID="Label3" />
                </ItemTemplate>
            </asp:TemplateColumn>
        </Columns>
    </asp:DataGrid>
</form>

```



```

        <asp:TemplateColumn HeaderText="tModel Key"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
            <ItemTemplate>
                <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "PID") %>' ID="Label4" />
            </ItemTemplate>
        </asp:TemplateColumn>
        <asp:TemplateColumn HeaderText="End-to-end Delay"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
            <ItemTemplate>
                <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "END2ENDDelay") %>' ID="Label6" />
            </ItemTemplate>
        </asp:TemplateColumn>
        <asp:TemplateColumn HeaderText="Availability"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
            <ItemTemplate>
                <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "AVAILABILITY") %>' ID="Label7" />
            </ItemTemplate>
        </asp:TemplateColumn>
        <asp:TemplateColumn HeaderText="Mobile?" HeaderStyle-
Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
            <ItemTemplate>
                <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "MOBILITY") %>' ID="Label8" />
            </ItemTemplate>
        </asp:TemplateColumn>
    </Columns>
</asp:DataGrid>

<!-- Catch no results to notify user -->
<p align="center">
    <asp:label id="DataGridIsNullLabel1" runat="server" Font-
Bold="true"></asp:label>
</p>

<!-- Display all desktop components found by HHs -->
<h4 align="center">Component Search Location #2 (Desktop)</h4>
<asp:DataGrid ID="DG_Components2" AutoGenerateColumns="False"
Width="99%" BorderColor="#000000" Runat="server">
    <HeaderStyle Font-Size="13px" Font-Families="Verdana" Font-
Bold="True" BackColor="#003366"></HeaderStyle>
    <ItemStyle Font-Size="13px" Font-Families="verdana"
BackColor="Beige"></ItemStyle>
    <Columns>
        <asp:TemplateColumn HeaderText="Web Service Access
Point" HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
            <ItemTemplate>
                <asp:Label id="Label9" Runat="server"
Font-Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "BINDING") %>' />
            </asp:Label>
        </ItemTemplate>
    </asp:TemplateColumn>
    <asp:TemplateColumn HeaderText="Service Client"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">

```

```

                                <ItemTemplate>
                                    <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "PROXY") %>' ID="Label10" />
                                </ItemTemplate>
                                </asp:TemplateColumn>
                                <asp:TemplateColumn HeaderText="Name" HeaderStyle-
Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                                    <ItemTemplate>
                                        <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "NAME") %>' ID="Label11" />
                                    </ItemTemplate>
                                </asp:TemplateColumn>
                                <asp:TemplateColumn HeaderText="Description"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                                    <ItemTemplate>
                                        <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "DESCRIPTION") %>' ID="Label12" />
                                    </ItemTemplate>
                                </asp:TemplateColumn>
                                <asp:TemplateColumn HeaderText="tModel Key"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                                    <ItemTemplate>
                                        <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "PID") %>' ID="Label13" />
                                    </ItemTemplate>
                                </asp:TemplateColumn>
                                <asp:TemplateColumn HeaderText="End-to-end Delay"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                                    <ItemTemplate>
                                        <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "END2ENDDELAY") %>' ID="Label14"
/>
                                    </ItemTemplate>
                                </asp:TemplateColumn>
                                <asp:TemplateColumn HeaderText="Availability"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                                    <ItemTemplate>
                                        <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "AVAILABILITY") %>' ID="Label15" />
                                    </ItemTemplate>
                                </asp:TemplateColumn>
                                <asp:TemplateColumn HeaderText="Mobile?" HeaderStyle-
Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                                    <ItemTemplate>
                                        <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "MOBILITY") %>' ID="Label16" />
                                    </ItemTemplate>
                                </asp:TemplateColumn>
                            </Columns>
                        </asp:DataGrid>

                        <!-- Catch no results to notify user -->
                        <p align="center">
                            <asp:label id="DataGridIsNullLabel2" runat="server" Font-
Bold="true"></asp:label>
                        </p>

```

```

        <!-- Display all embedded components found by HHs -->
        <h4 align="center">Component Search Location #3 (Embedded Device)</h4>
        <asp:DataGrid ID="DG_Components3" AutoGenerateColumns="False"
Width="99%" BorderColor="#000000" Runat="server">
            <HeaderStyle Font-Size="13px" Font-Names="Verdana" Font-
Bold="True" BackColor="#003366"></HeaderStyle>
            <ItemStyle Font-Size="13px" Font-Names="verdana"
BackColor="Beige"></ItemStyle>
            <Columns>
                <asp:TemplateColumn HeaderText="Web Service Access
Point" HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                    <ItemTemplate>
                        <asp:Label id="Label18" Runat="server"
Font-Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "BINDING") %>'>
                        </asp:Label>
                    </ItemTemplate>
                </asp:TemplateColumn>
                <asp:TemplateColumn HeaderText="Service Client"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                    <ItemTemplate>
                        <asp:Label Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "PROXY") %>' ID="Label19" />
                    </ItemTemplate>
                </asp:TemplateColumn>
                <asp:TemplateColumn HeaderText="Name" HeaderStyle-
Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                    <ItemTemplate>
                        <asp:Label Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "NAME") %>' ID="Label20" />
                    </ItemTemplate>
                </asp:TemplateColumn>
                <asp:TemplateColumn HeaderText="Description"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                    <ItemTemplate>
                        <asp:Label Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "DESCRIPTION") %>' ID="Label21" />
                    </ItemTemplate>
                </asp:TemplateColumn>
                <asp:TemplateColumn HeaderText="tModel Key"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                    <ItemTemplate>
                        <asp:Label Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "PID") %>' ID="Label22" />
                    </ItemTemplate>
                </asp:TemplateColumn>
                <asp:TemplateColumn HeaderText="End-to-end Delay"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                    <ItemTemplate>
                        <asp:Label Runat="server" Font-
Size="10px" Text='<%=# DataBinder.Eval(Container.DataItem, "END2ENDDELAY") %>' ID="Label23"
/>
                    </ItemTemplate>
                </asp:TemplateColumn>
                <asp:TemplateColumn HeaderText="Availability"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff">

```

```

                                <ItemTemplate>
                                    <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "AVAILABILITY") %>' ID="Label24" />
                                </ItemTemplate>
                                </asp:TemplateColumn>
                                <asp:TemplateColumn HeaderText="Mobile?" HeaderStyle-
Font-Bold="true" HeaderStyle-ForeColor="#ffffff">
                                    <ItemTemplate>
                                        <asp:Label Runat="server" Font-
Size="10px" Text='<%# DataBinder.Eval(Container.DataItem, "MOBILITY") %>' ID="Label25" />
                                    </ItemTemplate>
                                </asp:TemplateColumn>
                                </Columns>
                            </asp:DataGrid>

                            <!-- Catch no results to notify user -->
                            <p align="center">
                                <asp:label id="DataGridIsNullLabel3" runat="server" Font-
Bold="true"></asp:label>
                            </p>
                        </form>
                    </body>
</HTML>

```

Global.asax.vb

```

Imports System.Web
Imports System.Web.SessionState

Public Class Global
    Inherits System.Web.HttpApplication

    #Region " Component Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Component Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Required by the Component Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Component Designer
    'It can be modified using the Component Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

#End Region

```

```

Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when the application is started
End Sub

Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when the session is started
End Sub

Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires at the beginning of each request
End Sub

Sub Application_AuthenticateRequest(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires upon attempting to authenticate the use
End Sub

Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when an error occurs
End Sub

Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when the session ends
End Sub

Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when the application ends
End Sub

End Class

```

GetCustomerAccountsClient.aspx

```

<%@ Page Language="vb" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<!--
Filename:          GetCustomerAccountsClient.aspx
Created Date:      3/28/03
Author:            Bob Berbeco
Description:        This web page consumes the GetCustomerAccounts detail function
                    in the BankDataSvc web service using a web service proxy.
                    Results are displayed and editable.
Modifications:     4/13/03 Bob Berbeco - added comments
-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
    <HEAD>
        <LINK href="..\Styles.css" type="text/css" rel="stylesheet">
        <script runat="server">

            'Page load subroutine
            Public Sub Page_Load(Source As Object, E As EventArgs)
                If Not Page.IsPostBack Then
                    BindData()

```

```

        End If
    End Sub

    'BindGrid subroutine for data grid configuration
    'Use the web service proxy for retrieval
    Public Sub BindData()

        'Declaration format: Dim variablename As New Namespace.Class
        Dim wsProxy As New GetCustomerAccountsClient.BankDataService()
        customerAccts.DataSource = wsProxy.GetCustomerAccounts()
        customerAccts.DataBind()
    End Sub

    'Data grid edit subroutines - get item index and change data grid display
    Public Sub DataGrid1_Edit(ByVal Source As Object, _
    ByVal E As DataGridCommandEventArgs)
        customerAccts.EditItemIndex = E.Item.ItemIndex
        BindData()
    End Sub

    'Data grid cancel subroutines - change data grid display back to original
    Public Sub DataGrid1_Cancel(ByVal Source As Object, _
    ByVal E As DataGridCommandEventArgs)
        customerAccts.EditItemIndex = -1
        BindData()
    End Sub

    'Data grid update subroutines - update the data grid output into database
    Public Sub DataGrid1_Update(ByVal Source As Object, _
    ByVal E As DataGridCommandEventArgs)
        Dim myConnection As SqlConnection
        Dim myCommand As SqlCommand

        'Initiatialize all the updateable text for the update query
        Dim txtBalance As TextBox = E.Item.Cells(4).Controls(0)
        Dim strUpdateStmt As String

        'Update query
        strUpdateStmt = "UPDATE Customer_Accounts SET " & _
        "Balance = " & txtBalance.Text & " " & _
        "WHERE CustomerID = " & E.Item.Cells(1).Text & " " & _
        "AND AccountNumber = " & E.Item.Cells(2).Text & " "

        'Create connection to database and update table
        myConnection = New SqlConnection( _
        "User Id=HeadHunter_1;Password=HeadHunter1;database=bank;server=IN-
CANC-867126\MP")
        myCommand = New SqlCommand(strUpdateStmt, myConnection)
        myConnection.Open()
        myCommand.ExecuteNonQuery()
        customerAccts.EditItemIndex = -1
        BindData()
    End Sub
</script>
</HEAD>
<body>

```

```

<p align="left"><IMG height="96" src="..\gfx\uniframeheader.gif" width="520">
</p>
<H3>Get Customer Accounts Client</H3>
<p><i><b>Comments:</b><br>Client consumes the GetCustomerAccounts detail
function in the
BankDataSvc web service using a web service proxy.<br>Results are
displayed and editable.</i></p>
<form id="Form1" method="post" runat="server">
    <div id="queryDiv1"><asp:datagrid id="customerAccts" runat="server"
DataKeyField="CustomerID" PageSize="5" AutoGenerateColumns="False" Height="50px"
Width="100%" DataMember="CustomerInfos" CellPadding="2" OnEditCommand="DataGrid1_Edit"
OnCancelCommand="DataGrid1_Cancel" OnUpdateCommand="DataGrid1_Update">
        <HeaderStyle Font-Names="Verdana" Font-Bold="True"
Height="10px" ForeColor="White" BackColor="#003366"></HeaderStyle>
        <ItemStyle Font-Size="13px" Font-Names="verdana"
BackColor="Beige"></ItemStyle>
        <Columns>
            <asp:EditCommandColumn
ButtonType="Linkbutton" UpdateText="Update" CancelText="Cancel" EditText="Edit" HeaderStyle-
Font-Bold="true" HeaderStyle-ForeColor="#ffffff"></asp:EditCommandColumn>
            <asp:BoundColumn DataField="CustomerID"
ReadOnly="True" HeaderText="Account #" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
            <asp:BoundColumn DataField="AccountNumber"
ReadOnly="True" HeaderText="Account Number" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
            <asp:BoundColumn DataField="AccountType"
ReadOnly="True" HeaderText="Account Type" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
            <asp:BoundColumn DataField="Balance"
HeaderText="Balance" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
            <asp:BoundColumn
DataField="Customer_Firstname" ReadOnly="True" HeaderText="Account Holder Firstname"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff"></asp:BoundColumn>
            <asp:BoundColumn
DataField="Customer_Lastname" ReadOnly="True" HeaderText="Account Holder Lastname"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="#ffffff"></asp:BoundColumn>
        </Columns>
    </asp:datagrid></div>
</form>
</body>
</HTML>

```

GetInventoryAccountsClient.aspx

```

<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Data" %>
<%@ Page Language="vb" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
    <HEAD>
        <title>Get Inventory Accounts Client</title>
        <!--

```

Filename: GetInventoryAccountsClient.aspx

```

Created Date: 3/28/03
Author: Bob Berbeco
Description: This web page consumes the GetInventory detail function
              in the InventoryDataSvc web service using a web service proxy.
              Results are displayed and editable.
Modifications: 4/13/03 Bob Berbeco - added comments
-->
<LINK href="..\Styles.css" type="text/css" rel="stylesheet">
<script runat="server">

'Page load subroutine
Public Sub Page_Load(Source As Object, E As EventArgs)
    If Not Page.IsPostBack Then
        BindData()
    End If
End Sub

'BindGrid subroutine for data grid configuration
'Use the web service proxy for retrieval
Public Sub BindData()

    'Declaration format: Dim variablename As New Namespace.Class
    Dim wsProxy As New GetInventoryAccountsClient.InventoryDataService
    InventoryItems.DataSource = wsProxy.GetInventory()
    InventoryItems.DataBind()
End Sub

'Data grid edit subroutines - get item index and change data grid display
Public Sub DataGrid1_Edit(ByVal Source As Object, _
    ByVal E As DataGridCommandEventArgs)
    InventoryItems.EditItemIndex = E.Item.ItemIndex
    BindData()
End Sub

'Data grid cancel subroutines - change data grid display back to original
Public Sub DataGrid1_Cancel(ByVal Source As Object, _
    ByVal E As DataGridCommandEventArgs)
    InventoryItems.EditItemIndex = -1
    BindData()
End Sub

'Data grid update subroutines - update the data grid output into database
Public Sub DataGrid1_Update(ByVal Source As Object, _
    ByVal E As DataGridCommandEventArgs)
    Dim myConnection As SqlConnection
    Dim myCommand As SqlCommand

    'Initiatialize all the updateable text for the update query
    Dim txtQuantity As TextBox = E.Item.Cells(5).Controls(0)
    Dim strUpdateStmt As String

    'Update query
    strUpdateStmt = "UPDATE Inventory SET " & _
        "Quantity = '" & txtQuantity.Text & "' " & _
        "WHERE ID = '" & E.Item.Cells(1).Text & "'"

```



```

        'Create connection to database and update table
        myConnection = New SqlConnection( _
            "User
Id=HeadHunter_1;Password=HeadHunter1;database=manufacturing;server=IN-CANC-867126\MP")
        myCommand = New SqlCommand(strUpdateStmt, myConnection)
        myConnection.Open()
        myCommand.ExecuteNonQuery()
        InventoryItems.EditItemIndex = -1
        BindData()
    End Sub
</script>
</HEAD>
<body>
    <p align="left"><IMG height="96" src="..\gfx\uniframeheader.gif" width="520">
    </p>
    <H3>Get Inventory Accounts Client</H3>
    <p><i><b>Comments:</b><br>
        Client consumes the GetInventory detail function in the
InventoryDataSvc web
        service using a web service proxy.<br>
        Results are displayed and editable.</i></p>
    <form id="Form1" method="post" runat="server">
        <div id="queryDiv1"><asp:datagrid id="InventoryItems" runat="server"
DataKeyField="ID" PageSize="5" AutoGenerateColumns="False" Height="50px" Width="100%"
DataMember="InventoryItems" CellPadding="2" OnEditCommand="DataGrid1_Edit"
OnCancelCommand="DataGrid1_Cancel" OnUpdateCommand="DataGrid1_Update">
            <HeaderStyle Font-Names="Verdana" Font-Bold="True"
Height="10px" ForeColor="White" BackColor="#003366"></HeaderStyle>
            <ItemStyle Font-Size="13px" Font-Names="verdana"
BackColor="Beige"></ItemStyle>
            <Columns>
                <asp:EditCommandColumn
ButtonType="Linkbutton" UpdateText="Update" CancelText="Cancel" EditText="Edit" HeaderStyle-
Font-Bold="true" HeaderStyle-ForeColor="#ffffff"></asp:EditCommandColumn>
                <asp:BoundColumn DataField="ID"
ReadOnly="True" HeaderText="Inventory #" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
                <asp:BoundColumn DataField="Description"
ReadOnly="True" HeaderText="Description" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
                <asp:BoundColumn DataField="Cost"
ReadOnly="True" HeaderText="Cost" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
                <asp:BoundColumn DataField="Retail"
ReadOnly="True" HeaderText="Retail" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
                <asp:BoundColumn DataField="Quantity"
HeaderText="Quantity" HeaderStyle-Font-Bold="true" HeaderStyle-
ForeColor="#ffffff"></asp:BoundColumn>
            </Columns>
        </asp:datagrid></div>
    </form>
</body>
</HTML>

```

MyProxyClass.vb

```
Option Strict Off
Option Explicit On
```

```
Imports System
Imports System.ComponentModel
Imports System.Diagnostics
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization
```

```
'
'Saturn Web Development Tool
'
```

```
Namespace GetCustomerAccountsClient
```

```
    '<remarks/>
    <System.Diagnostics.DebuggerStepThroughAttribute(), _
        System.ComponentModel.DesignerCategoryAttribute("code"), _
        System.Web.Services.WebServiceBindingAttribute(Name:="BankDataServiceSoap",
[Namespace]:="http://localhost/rberbeco/UniFrame/BankDataSvc")> _
    Public Class BankDataService
        Inherits System.Web.Services.Protocols.SoapHttpClientProtocol
```

```
        '<remarks/>
        Public Sub New()
            MyBase.New
            Me.Url = "http://localhost/rberbeco/UniFrame/BankDataSvc/BankDataSvc.asmx"
        End Sub
```

```
        '<remarks/>
```

```
    <System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://localhost/rberbeco/UniFrame/Ban
kDataSvc/GetCustomerAccounts",
RequestNamespace:="http://localhost/rberbeco/UniFrame/BankDataSvc",
ResponseNamespace:="http://localhost/rberbeco/UniFrame/BankDataSvc",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
        Public Function GetCustomerAccounts() As System.Data.DataSet
            Dim results() As Object = Me.Invoke("GetCustomerAccounts", New Object(-1) {})
            Return CType(results(0),System.Data.DataSet)
        End Function
```

```
        '<remarks/>
        Public Function BeginGetCustomerAccounts(ByVal callback As System.AsyncCallback, ByVal
asyncState As Object) As System.IAsyncResult
            Return Me.BeginInvoke("GetCustomerAccounts", New Object(-1) {}, callback, asyncState)
        End Function
```

```
        '<remarks/>
        Public Function EndGetCustomerAccounts(ByVal asyncResult As System.IAsyncResult) As
System.Data.DataSet
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),System.Data.DataSet)
        End Function
```

```

'<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://localhost/rberbeco/UniFrame/InventoryDataSvc/GetInventory",
RequestNamespace:="http://localhost/rberbeco/UniFrame/InventoryDataSvc",
ResponseNamespace:="http://localhost/rberbeco/UniFrame/InventoryDataSvc",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
    Public Function GetInventory() As System.Data.DataSet
        Dim results() As Object = Me.Invoke("GetInventory", New Object(-1) {})
        Return CType(results(0),System.Data.DataSet)
    End Function

'<remarks/>
    Public Function BeginGetInventory(ByVal callback As System.AsyncCallback, ByVal asyncState As
Object) As System.IAsyncResult
        Return Me.BeginInvoke("GetInventory", New Object(-1) {}, callback, asyncState)
    End Function

'<remarks/>
    Public Function EndGetInventory(ByVal asyncResult As System.IAsyncResult) As
System.Data.DataSet
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),System.Data.DataSet)
    End Function

'<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://localhost/rberbeco/UniFrame/BankDataSvc/UpdateCustomerAccounts",
RequestNamespace:="http://localhost/rberbeco/UniFrame/BankDataSvc",
ResponseNamespace:="http://localhost/rberbeco/UniFrame/BankDataSvc",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
    Public Function UpdateCustomerAccounts() As System.Data.DataSet
        Dim results() As Object = Me.Invoke("UpdateCustomerAccounts", New Object(-1) {})
        Return CType(results(0),System.Data.DataSet)
    End Function

'<remarks/>
    Public Function BeginUpdateCustomerAccounts(ByVal callback As System.AsyncCallback, ByVal
asyncState As Object) As System.IAsyncResult
        Return Me.BeginInvoke("UpdateCustomerAccounts", New Object(-1) {}, callback, asyncState)
    End Function

'<remarks/>
    Public Function EndUpdateCustomerAccounts(ByVal asyncResult As System.IAsyncResult) As
System.Data.DataSet
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),System.Data.DataSet)
    End Function
End Class
End Namespace

```

MyProxyClass2.vb

```
Option Strict Off
Option Explicit On
```

```
Imports System
Imports System.ComponentModel
Imports System.Diagnostics
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization
```

```
,
'Saturn Web Development Tool
,
```

```
Namespace GetInventoryAccountsClient
```

```
    '<remarks/>
    <System.Diagnostics.DebuggerStepThroughAttribute(), _
    System.ComponentModel.DesignerCategoryAttribute("code"), _
    System.Web.Services.WebServiceBindingAttribute(Name:="InventoryDataServiceSoap",
[Namespace]:="http://localhost/rberbeco/UniFrame/InventoryDataSvc")> _
    Public Class InventoryDataService
        Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

        '<remarks/>
        Public Sub New()
            MyBase.New
            Me.Url = "http://localhost/rberbeco/UniFrame/WebServices/InventoryDataSvc_1.asmx"
        End Sub

        '<remarks/>
```

```
    <System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://localhost/rberbeco/UniFrame/InventoryDataSvc/GetInventory",
    RequestNamespace:="http://localhost/rberbeco/UniFrame/InventoryDataSvc",
    ResponseNamespace:="http://localhost/rberbeco/UniFrame/InventoryDataSvc",
    Use:=System.Web.Services.Description.SoapBindingUse.Literal,
    ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
        Public Function GetInventory() As System.Data.DataSet
            Dim results() As Object = Me.Invoke("GetInventory", New Object(-1) {})
            Return CType(results(0),System.Data.DataSet)
        End Function
```

```
        '<remarks/>
        Public Function BeginGetInventory(ByVal callback As System.AsyncCallback, ByVal asyncState As
Object) As System.IAsyncResult
            Return Me.BeginInvoke("GetInventory", New Object(-1) {}, callback, asyncState)
        End Function
```

```
        '<remarks/>
        Public Function EndGetInventory(ByVal asyncResult As System.IAsyncResult) As
System.Data.DataSet
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),System.Data.DataSet)
        End Function
```

End Class End Namespace
